China-pub.com
下载

第一部分　shell

第1章　文件安全与权限

为了防止未授权用户访问你的文件，可以在文件和目录上设置权限位。还可以设定文件在创建时所具有的缺省权限：这些只是整个系统安全问题中的一小部分。在这里我们并不想对系统安全问题的方方面面进行全面的探讨，只是介绍一下有关文件和目录的安全问题。

本章包含以下内容：

• 文件和目录的权限。

• setuid。

• chown和chgrp。

• umask。

• 符号链接。

创建文件的用户和他(她)所属于的组拥有该文件。文件的属主可以设定谁具有读、写、执行该文件的权限。当然，根用户或系统管理员可以改变任何普通用户的设置。一个文件一经创建，就具有三种访问方式：

1) 读，可以显示该文件的内容。

2) 写，可以编辑或删除它。

3) 执行，如果该文件是一个shell脚本或程序。

按照所针对的用户，文件的权限可分为三类：

1) 文件属主，创建该文件的用户。

2) 同组用户，拥有该文件的用户组中的任何用户。

3) 其他用户，即不属于拥有该文件的用户组的某一用户。

1.1　文件

当你创建一个文件的时候，系统保存了有关该文件的全部信息，包括：

• 文件的位置。

• 文件类型。

• 文件长度。

• 哪位用户拥有该文件，哪些用户可以访问该文件。

• i节点。

• 文件的修改时间。

• 文件的权限位。

让我们使用ls -l命令，来看一个典型的文件：

China-pub.com 下载

```
$ ls -l
total 4232
-rwxr-xr-x   1 root   root      3756  Oct 14 04:44 dmesg
-r-xr-xr-x   1 root   root      12708 Oct  3 05:40 ps
-rwxr-xr-x   1 root   root      5388  Aug  5  1998 pwd
....
```

下面让我们来分析一下该命令所得结果的前面两行，看看都包含了哪些信息：

total 4232：这一行告诉我们该目录中所有文件所占的空间。

-rwxr-xr-x：这是该文件的权限位。如果除去最前面的横杠，这里一共是 9个字符，他们分别对应9个权限位。通过这些权限位，可以设定用户对文件的访问权限。这 9个字符可以分为三组：

rwx：文件属主权限　　这是前面三位

r-x：同组用户权限　　这是中间三位

r-x：其他用户权限　　这是最后三位

后面我们还将对这些权限位作更详细的介绍。出现在 r、w、x位置上的横杠表示相应的访问权限被禁止。

1　该文件硬链接的数目。

root　文件的属主。

root　文件的属主root所在的缺省组(也叫做root)。

3578　用字节来表示的文件长度，记住，不是 K字节！

Oct 14 04:44　文件的更新时间。

dmesg　文件名。

1.2 文件类型

还记得前面一节所提到的文件权限位前面的那个字符吗？我们现在就解释一下这个横杠所代表的意思，文件类型有七种，它可以从 ls -l命令所列出的结果的第一位看出，这七种类型是：

d　目录。

l　符号链接(指向另一个文件)。

s　套接字文件。

b　块设备文件。

c　字符设备文件。

p　命名管道文件。

- 普通文件，或者更准确地说，不属于以上几种类型的文件。

1.3 权限

让我们用touch命令创建一个文件：

```
$ touch myfile
```

现在对该目录使用ls -l命令：

```
$ ls -l
-rw-r--r--   1 dave    admin          0 Feb 19 22:05 myfile
```

China-pub.com
下载

我们已经创建了一个空文件，正如我们所希望的那样，第一个横杠告诉我们该文件是一个普通文件。你将会发现所创建的文件绝大多数都是普通文件或符号链接文件 (后面将会出现更多的符号链接文件)。

文件属主权限	组用户权限	其他用户权限
rw-	r--	r—

接下来的三个权限位是文件属主所具有的权限；再接下来的三位是与你同组用户所具有的权限，这里是admin组；最后三位是其他用户所具有的权限。在该命令的结果中，我所属于的缺省组也显示了出来。下面是对该文件权限的精确描述：

表1-1　ls -l命令输出的含义

(第一个字符)-	普通文件
(接下来的三个字符)rw-	文件属主的权限
(再接下来的三个字符)r--	同组用户的权限
(最后三个字符)r--	其他用户的权限

因此，这三组字符(除了第一个字符)分别定义了：

1) 文件属主所拥有的权限。

2) 文件属主缺省组(一个用户可以属于很多的组)所拥有的权限。

3) 系统中其他用户的权限。

在每一组字符中含有三个权限位：

r　读权限

w　写/更改权限

x　执行该脚本或程序的权限

这里我们采用另外一种方式来表示刚才所列出 myfile的文件权限：

-	rw-	r--	r--
文件类型为普通文件	文件属主可以读、写	同组用户可以读	其他用户可以读

你可能已经注意到了，myfile在创建的时候并未给属主赋予执行权限，在用户创建文件时，系统不会自动地设置执行权限位。这是出于加强系统安全的考虑。必须手工修改这一权限位：后面讲到umask命令时，你就会明白为什么没有获得执行权限。然而，你可以针对目录设置执行权限位，但这与文件执行权限位的意义有所不同，这一点我们将在后面讨论。

上面这段关于权限位的内容可能不太好理解，让我们来看几个例子 (见表1-2)。

更令人迷惑的是，对于文件属主来说，在只有读权限位被置位的情况下，仍然可以通过文件重定向的方法向该文件写入。过一会儿我们就会看到，能否删除一个文件还依赖于该文件所在目录权限位的设置。

表1-2　文件权限及含义

权　限	所代表的含义
r-- --- ---	文件属主可读，但不能写或执行
r-- r-- ---	文件属主和同组用户(一般来说，是文件属主所在的缺省组)可读
r-- r-- r--	任何用户都可读，但不能写或执行
rwx r-- r--	文件属主可读、写、执行，同组用户和其他用户只可读
rwx r-x ---	文件属主可读、写、执行，同组用户可读、执行

China-pub.com 下载

(续)

权　限	所代表的含义
rwx r-x r-x	文件属主可读、写、执行，同组用户和其他用户可读、执行
rw- rw- ---	文件属主和同组用户可读、写
rw- rw- r--	文件属主和同组用户可读、写，其他用户可读
rw- rw- ---	文件属主和同组用户及其他用户读可以读、写，慎用这种权限设置，因为任何用户都可以写入该文件

1.4 改变权限位

对于属于你的文件，可以按照自己的需要改变其权限位的设置。在改变文件权限位设置之前，要仔细地想一想有哪些用户需要访问你的文件(包括你的目录)。可以使用chmod命令来改变文件权限位的设置。这一命令有比较短的绝对模式和长一些的符号模式。我们先来看一看符号模式。

1.4.1 符号模式

chmod命令的一般格式为：

```
chmod [who] operator [permission] filename
```

who的含义是：

u 文件属主权限。

g 同组用户权限。

o 其他用户权限。

a 所有用户(文件属主、同组用户及其他用户)。

operator的含义：

\+ 增加权限。

\- 取消权限。

= 设定权限。

permission的含义：

r 读权限。

w 写权限。

x 执行权限。

s 文件属主和组set-ID。

t 粘性位*。

l 给文件加锁，使其他用户无法访问。

u,g,o 针对文件属主、同组用户及其他用户的操作。

*在列文件或目录时，有时会遇到“ t ”位。“ t ”代表了粘性位。如果在一个目录上出现“ t ”位，这就意味着该目录中的文件只有其属主才可以删除，即使某个同组用户具有和属主同等的权限。不过有的系统在这一规则上并不十分严格。

如果在文件列表时看到“ t ”，那么这就意味着该脚本或程序在执行时会被放在交换区(虚存)。不过由于当今的内存价格如此之低，大可不必理会文件的“ t ”的使用。

China-pub.com 下载

1.4.2 chmod命令举例

现在让我们来看一些使用chmod命令的例子。假定myfile文件最初具有这样的权限：rwx rwx rwx ：

命 令	结 果	含 义
chmod a-x myfile	rw- rw- rw-	收回所有用户的执行权限
chmod og-w myfile	rw- r-- r--	收回同组用户和其他用户的写权限
chmod g+w myfile	rw- rw- r--	赋予同组用户写权限
chmod u+x myfile	rwx rw- r--	赋予文件属主执行权限
chmod go+x myfile	rwx rwx r-x	赋予同组用户和其他用户执行权限

当创建myfile文件时，它具有这样的权限：

-rw-r--r-- 1 dave admin 0 Feb 19 22:05 myfile

如果这是我写的一个脚本，我希望能够具有执行权限，并取消其他用户 (所有其他用户)的写权限，可以用：

```
$ chmod u+x o-w myfile
```

这样，该文件的权限变为：

-rwx r-- --- 1 dave admin 0 Feb 19 22:05 myfile

现在已经使文件属主对myfile文件具有读、写执行的权限，而admin组的用户对该文件具有读权限。

如果希望某个脚本文件对你自己来说可执行，而且你对该文件的缺省权限很放心，那么只要使它对你来说具有执行权限即可。

```
$ chmod u+x dt
```

1.4.3 绝对模式

chmod命令绝对模式的一般形式为：

```
chmod [mode] file
```

其中mode是一个八进制数。

在绝对模式中，权限部分有着不同的含义。每一个权限位用一个八进制数来代表，如表1-3所示。

表1-3 八进制目录/文件权限表示

八 进 制 数	含 义	八 进 制 数	含 义
0400	文件属主可读	0010	同组用户可执行
0200	文件属主可写	0004	其他用户可读
0100	文件属主可执行	0002	其他用户可写
0040	同组用户可读	0001	其他用户可执行
0020	同组用户可写		

在设定权限的时候，只需按照表1-3查出与文件属主、同组用户和其他用户所具有的权限相对应的数字，并把它们加起来，就是相应的权限表示。

从表1-3中可以看出，文件属主、同组用户和其他用户分别所能够具有的最大权限值就是7。

China-pub.com 下载

再来看看前面举的例子：

-rw-r--r-- 1 dave admin 0 Feb 19 22:05 myfile

相应的权限表示应为644，它的意思就是：

0400+0200(文件属主可读、写) =0600
0040(同组用户可读) =0040
0004(其他用户可读) =0004
0644

有一个计算八进制权限表示的更好办法，如表1-4所示：

表1-4 计算权限值

文件属主	同组用户	其他用户
r w x 4 + 2 + 1	r w x 4 + 2 + 1	r w x 4 + 2 + 1

使用表1-4，可以更容易地计算出相应的权限值，只要分别针对文件属主、同组用户和其他用户把相应权限下面的数字加在一起就可以了。

myfile文件具有这样的权限：

r w - r - - r - -
4 + 2 4 4

把相应权限位所对应的值加在一起，就是644。

1.4.4 chmod命令的其他例子

以下是一些chmod命令绝对模式的例子：

命 令	结 果	含 义
chmod 666	rw- rw- rw-	赋予所有用户读和写的权限
chmod 644	rw- r-- r--	赋予所有文件属主读和写的权限，所有其他用户读权限
chmod 744	rwx r-- r--	赋予文件属主读、写和执行的权限，所有其他用户读的权限
chmod 664	rw- rw- r--	赋予文件属主和同组用户读和写的权限，其他用户读权限
chmod 700	rwx --- ---	赋予文件属主读、写和执行的权限
chmod 444	r-- r-- r--	赋予所有用户读权限

下面举一个例子，假定有一个名为yoa的文件，具有如下权限：

-rw-rw-r-- 1 dave admin 455 Feb 19 22:05 yoa

我现在希望使自己对该文件可读、写和执行， admin组用户对该文件只读，可以键入：

```
$ chmod 740 yoa
-rwxr-- ---    1 dave    admin          455 Feb 19 22:05 yoa
```

如果希望自己对该文件可读、写和执行，对其他所有用户只读，我可以用：

```
$ chmod 744 myfile
-rwxr-- r--    1 dave    admin            0 Feb 19 22:05 myfile
```

如果希望一次设置目录下所有文件的权限，可以用：

```
chmod 644*
```

这将使文件属主和同组用户都具有读和写的权限，其他用户只具有读权限。

China-pub.com
下载

还可以通过使用-R选项连同子目录下的文件一起设置：

```
chmod -R 664 /usr/local/home/dave/*
```

这样就可以一次将/usr/local/home/dave目录下的所有文件连同各个子目录下的文件的权限全部设置为文件属主和同组用户可读和写，其他用户只读。使用 -R选项一定要谨慎，只有在需要改变目录树下全部文件权限时才可以使用。

1.4.5　可以选择使用符号模式或绝对模式

上面的例子中既有绝对模式的，也有符号模式的，我们可以从中看出，如果使用该命令的符号模式，可以设置或取消个别权限位，而在绝对模式中则不然。我个人倾向于使用符号模式，因为它比绝对模式方便快捷。

1.5　目录

还记得在前面介绍chmod命令时讲过，目录的权限位和文件有所不同。现在我们来看看其中的区别。目录的读权限位意味着可以列出其中的内容。写权限位意味着可以在该目录中创建文件，如果不希望其他用户在你的目录中创建文件，可以取消相应的写权限位。执行权限位则意味着搜索和访问该目录（见表 1-5、表1-6）。

表1-5　目录权限

r	w	x
可以列出该目录中的文件	可以在该目录中创建或删除文件	可以搜索或进入该目录

表1-6　目录权限举例

权　　限	文件属主	同组用户	其他用户
drwx rwx r-x(775)	读、写、执行	读、写、执行	读、执行
drwx r-x r--(754)	读、写、执行	读、执行	读
drwx r-x r-x(755)	读、写、执行	读、执行	读、执行

如果把同组用户或其他用户针对某一目录的权限设置为 --x，那么他们将无法列出该目录中的文件。如果该目录中有一个执行位置位的脚本或程序，只要用户知道它的路径和文件名，仍然可以执行它。用户不能够进入该目录并不妨碍他的执行。

目录的权限将会覆盖该目录中文件的权限。例如，如果目录 docs具有如下的权限：

```
drwx r-- r--  1  louise  admin      2390  Jul 23  09:44 docs
```

而其中的文件pay的权限为：

```
-rwx rwx rwx  1  louise   admin      5567 Oct  3 05:40 pay
```

那么admin组的用户将无法编辑该文件，因为它所属的目录不具有这样的权限。

该文件对任何用户都可读，但由于它所在的目录并未给 admin组的用户赋予执行权限，所以该组的用户都将无法访问该目录，他们将会得到“访问受限”的错误消息。

1.6　suid/guid

我们在前面曾经提到过suid和guid。这种权限位近年来成为一个棘手的问题。很多系统供

China-pub.com 下载

应商不允许实现这一位，或者即使它被置位，也完全忽略它的存在，因为它会带来安全性风险。那么人们为何如此大惊小怪呢？

suid意味着如果某个用户对属于自己的shell脚本设置了这种权限，那么其他用户在执行这一脚本时也会具有其属主的相应权限。于是，如果根用户的某一个脚本设置了这样的权限，那么其他普通用户在执行它的期间也同样具有根用户的权限。同样的原则也适用于 guid，执行相应脚本的用户将具有该文件所属用户组中用户的权限。

1.6.1 为什么要使用suid/guid

为什么要使用这种类型的脚本？这里有一个很好的例子。我管理着几个大型的数据库系统，而对它们进行备份需要有系统管理权限。我写了几个脚本，并设置了它们的 guid，这样我指定的一些用户只要执行这些脚本就能够完成相应的工作，而无须以数据库管理员的身份登录，以免不小心破坏了数据库服务器。通过执行这些脚本，他们可以完成数据库备份及其他管理任务，但是在这些脚本运行结束之后，他们就又回复到他们作为普通用户的权限。

有相当一些UNIX命令也设置了suid和guid。如果想找出这些命令，可以进入/bin或/sbin目录，执行下面的命令：

```
$ ls -l | grep '^...s'
```

上面的命令是用来查找suid文件的；

```
$ ls -l | grep '^...s..s'
```

上面的命令是用来查找suid和guid的。

现在我们明白了什么是suid，可是如何设置它呢？下面就来介绍这个问题。如果希望设置suid，那么就将相应的权限位之前的那一位设置为4；如果希望设置guid，那么就将相应的权限位之前的那一位设置为2；如果希望两者都置位，那么将相应的权限位之前的那一位设置为4+2。

一旦设置了这一位，一个s将出现在x的位置上。记住：在设置suid或guid的同时，相应的执行权限位必须要被设置。例如，如果希望设置 guid，那么必须要让该用户组具有执行权限。

如果想要对文件login设置suid，它当前所具有的权限为 rwx rw- r-- (741)，需要在使用chmod命令时在该权限数字的前面加上一个 4，即chmod 4741，这将使该文件的权限变为 rws rw- r--。

```
$ chmod 4741 logit
```

1.6.2 设置suid/guid的例子

下面给出几个例子：

表1-7 设置suid/guid

命 令	结 果	含 义
chmod 4755	rws r-x r-x	文件被设置了suid，文件属主具有读、写和执行的权限，所有其他用户具有读和执行的权限
chmod 6711	rws --s --s	文件被设置了suid和guid，文件属主具有读、写和执行的权限，所有其他用户具有执行的权限
chmod 4764	rws rw- r--	文件被设置了suid，文件属主具有读、写和执行的权限，同组用户具有读和执行的权限，其他用户具有读权限

China-pub.com 下载

还可以使用符号方式来设置suid/guid。如果某个文件具有这样的权限：rwx r-x r-x，那么可以这样设置其suid：

```
chmod u+s <filename>
```

于是该文件的权限将变为：rws r-x r-x

在查找设置了suid的文件时，没准会看到具有这样权限的文件：rwS r-x r-x，其中S为大写。它表示相应的执行权限位并未被设置，这是一种没有什么用处的suid设置，可以忽略它的存在。

注意，chmod命令不进行必要的完整性检查，可以给某一个没用的文件赋予任何权限，但chmod 命令并不会对所设置的权限组合做什么检查。因此，不要看到一个文件具有执行权限，就认为它一定是一个程序或脚本。

1.7 chown和chgrp

当你创建一个文件时，你就是该文件的属主。一旦你拥有某个文件，就可以改变它的所有权，把它的所有权交给另外一个 /etc/passwd文件中存在的合法用户。可以使用用户名或用户ID号来完成这一操作。在改变一个文件的所有权时，相应的 suid也将被清除，这是出于安全性的考虑。只有文件的属主和系统管理员可以改变文件的所有权。一旦将文件的所有权交给另外一个用户，就无法再重新收回它的所有权。如果真的需要这样做，那么就只有求助于系统管理员了。

chown命令的一般形式为：

```
chmod -R -h owner file
```

-R选项意味着对所有子目录下的文件也都进行同样的操作。 -h选项意味着在改变符号链接文件的属主时不影响该链接所指向的目标文件。

1.7.1 chown举例

这里给出几个例子：

```
$ ls -l
$ -rwxrwxrwx    1     louise    admin 345 Sep 20 14:33 project
$ chown pauline project
$ ls -l
$ -rwxrwxrwx    1     pauline   admin     Sep 20 14:33 project
```

文件project的所有权现在由用户louise交给了用户pauline。

1.7.2 chgrp举例

chgrp命令和chown命令的格式差不多，下面给出一个例子。

```
-rwxrwxrwx    1    pauline   admin    345 Sep 20 14:33 project
$ chgrp sybadmin project
$ ls -l
$ -rwxrwxrwx 1 pauline   sybadmin 345 Sep 20 14:33 project
```

用户pauline现在把该文件所属的组由admin变为sybadmin（系统中的另外一个用户组）。

1.7.3 找出你所属于的用户组

如果你希望知道自己属于哪些用户组，可以用如下的命令：

China-pub.com 下载

```
$ group
$ admin sysadmin appsgen general
```

或者可以使用id命令：

```
$ id
$ uid=0(root) gid=0(root)
groups=0(root),1(bin),2(daemon),3(sys),4(adm)
```

1.7.4 找出其他用户所属于的组

为了找出其他用户所属于的组，可以用如下的命令：

```
$ group matty
$ sybadmin appsgen post
```

上面的命令告诉我们用户matty属于sybadmin、appsgen和post用户组。

1.8 umask

当最初登录到系统中时， umask命令确定了你创建文件的缺省模式。这一命令实际上和chmod命令正好相反。你的系统管理员必须要为你设置一个合理的 umask值，以确保你创建的文件具有所希望的缺省权限，防止其他非同组用户对你的文件具有写权限。

在已经登录之后，可以按照个人的偏好使用 umask命令来改变文件创建的缺省权限。相应的改变直到退出该shell或使用另外的umask命令之前一直有效。

一般来说， umask命令是在/etc/profile文件中设置的，每个用户在登录时都会引用这个文件，所以如果希望改变所有用户的 umask，可以在该文件中加入相应的条目。如果希望永久性地设置自己的umask值，那么就把它放在自己 $HOME目录下的.profile或.bash_profile文件中。

1.8.1 如何计算umask值

umask命令允许你设定文件创建时的缺省模式，对应每一类用户 (文件属主、同组用户、其他用户)存在一个相应的umask值中的数字。对于文件来说，这一数字的最大值分别是 6。系统不允许你在创建一个文本文件时就赋予它执行权限，必须在创建后用 chmod命令增加这一权限。目录则允许设置执行权限，这样针对目录来说， umask中各个数字最大可以到7。

该命令的一般形式为：

```
umask nnn
```

其中nnn为umask置000-777。

让我们来看一些例子。

计算出你的umask值：

可以有几种计算umask值的方法，通过设置umask值，可以为新创建的文件和目录设置缺省权限。表1-8列出了与权限位相对应的umask值。

在计算umask值时，可以针对各类用户分别在这张表中按照所需要的文件 /目录创建缺省权限查找对应的umask值。

例如， umask值002 所对应的文件和目录创建缺省权限分别为 664和775。

还有另外一种计算umask值的方法。我们只要记住umask是从权限中“拿走”相应的位即可。

China-pub.com 下载

表1-8 umask值与权限

umask	文 件	目 录
0	6	7
1	6	6
2	4	5
3	4	4
4	2	3
5	2	2
6	0	1
7	0	0

例如，对于umask值002，相应的文件和目录缺省创建权限是什么呢？

第一步，我们首先写下具有全部权限的模式，即777(所有用户都具有读、写和执行权限)。

第二步，在下面一行按照umask值写下相应的位，在本例中是002。

第三步，在接下来的一行中记下上面两行中没有匹配的位。这就是目录的缺省创建权限。稍加练习就能够记住这种方法。

第四步，对于文件来说，在创建时不能具有文件权限，只要拿掉相应的执行权限比特即可。

这就是上面的例子，其中umask值为002：

1) 文件的最大权限	rwx rwx rwx (777)	
2) umask值为002	- - - - - - - -w-	
3) 目录权限	rwx rwx r-x (775)	这就是目录创建缺省权限
4) 文件权限	rw- rw- r-- (664)	这就是文件创建缺省权限

下面是另外一个例子，假设这次umask值为022：

1) 文件的最大权限	rwx rwx rwx (777)	
2) umask值为022	- - - -w- -w-	
3) 目录权限	rwx r-x r-x (755)	这就是目录创建缺省权限
4) 文件权限	rw- r-- r-- (644)	这就是文件创建缺省权限

1.8.2 常用的umask值

表1-9列出了一些umask值及它们所对应的目录和文件权限。

表1-9 常用的umask值及对应的文件和目录权限

umask值	目 录	文 件
022	755	644
027	750	640
002	775	664
006	771	660
007	770	660

如果想知道当前的umask 值，可以使用umask命令：

```
$ umask
$ 022
```

China-pub.com 下载

```
$ touch file1
$ ls -l file1
$ -rw-r--r--   1 dave     admin           0 Feb 18 42:05 file1
```

如果想要改变umask值，只要使用umask命令设置一个新的值即可：

```
$ umask 002
```

确认一下系统是否已经接受了新的umask值：

```
$ umask
$ 002
$ touch file2
$ ls -l file2
$ -rw-rw-r--   1 dave     admin           0 Feb 18 45:07 file2
```

在使用umask命令之前一定要弄清楚到底希望具有什么样的文件 /目录创建缺省权限。否则可能会得到一些非常奇怪的结果；例如，如果将 umask值设置为600，那么所创建的文件/目录的缺省权限就是066！

1.9 符号链接

存在两种不同类型的链接，软链接和硬链接，这里我们只讨论软链接。软链接实际上就是一个指向文件的指针。你将会发现这种软链接使用起来非常方便。

1.9.1 使用软链接来保存文件的多个映像

下面我们就解释一下符号链接是怎么回事。比方说在 /usr/local/admin/sales目录下有一个含有销售信息的文件，销售部门的每一个人都想看这份文件。你可以在每一位用户的 $HOME目录下建立一个指向该文件的链接，而不是在每个目录下拷贝一份。这样当需要更改这一文件时，只需改变一个源文件即可。每个销售 $HOME目录中的链接可以起任何名字，不必和源文件一致。

如果有很多子目录，而进入这些目录很费时间，在这种情况下链接也非常有用。可以针对$HOME目录下的一个很深的子目录创建一个链接。还有，比如在安装一个应用程序时，它的日志被保存到/usr/opt/app/log目录下，如果想把它保存在另外一个你认为更方便目录下，可以建立一个指向该目录的链接。

该命令的一般形式为：

```
ln [-s] source_path target_path
```

其中的路径可以是目录也可以是文件。让我们来看几个例子。

1.9.2 符号链接举例

假如系统中有40个销售和管理用户，销售用户使用一个销售应用程序，而管理用户使用一个管理应用程序。我作为系统管理员该怎么做呢？首先删除它们各自 $HOME目录下的所有.profile文件。然后在/usr/local/menus/目录下创建两个profile文件，一个是sales.profile，一个是admin.profile，它们分别为销售和管理人员提供了所需的环境，并引导他们进入相应的应用程序。现在我在所有销售人员的 $HOME目录下分别创建一个指向sales.profile的链接，在所有管理人员的 $HOME目录下分别创建一个指向 admin.profile文件的链接。注意，不必在上面命令格式中的target_path端创建相应文件，如果不存在这样一个文件，ln命令会自动创建该文

件。下面就是我对销售人员matty所做的操作。

```
$ cd /home/sales/matty
$ rm.profile
$ ln -s /usr/local/menus/sales.profile. profile
$ ls -al.profile
$ lrwx rwx  rwx   1  sales    admin 5567 Oct  3 05:40.profile->
/usr/local/menus/sales.profile
```

(你所看到的可能会与此稍有差别)。

这就是我所要做的全部工作；对于管理人员也是如此。而且如果需要作任何修改的话，只要改变销售和管理人员的profile文件即可，而不必对40个用户逐一进行修改。

下面是另外一个例子。我所管理的系统中有一个网络监视器，它将日志写在 /usr/opt/monitor/regstar目录下，但其他所有的日志都保存在 /var/adm/logs目录下，这样只需在该目录下建立一个指向原有文件的链接就可以在一个地方看所有的日志了，而不必花费很多时间分别进入各个相应的目录。下面就是所用的链接命令：

```
$ ln -s /usr/opt/monitor/regstar/reg.log /var/adm/logs/monitor.log
```

如果链接太多的话，可以删掉一些，不过切记不要删除源文件。

不管是否在同一个文件系统中，都可以创建链接。在创建链接的时候，不要忘记在原有目录设置执行权限。链接一旦创建，链接目录将具有权限 777或rwx rwx rwx，但是实际的原有文件的权限并未改变。

在新安装的系统上，通常要进行这样的操作，在/var目录中创建一个指向/tmp目录的链接，因为有些应用程序认为存在 /var/tmp目录(然而它实际上并不存在)，有些应用程序在该目录中保存一些临时文件。为了使所有的临时文件都放在一个地方，可以使用 ln命令在/var目录下建立一个指向/tmp目录的链接。

```
$ pwd /var
$ ln -s /tmp  /var/tmp
```

现在如果我在/var目录中列文件，就能够看到刚才建立的链接：

```
$ ls -l
$ lrwx rwx  rwx   1  root    root      5567 Sep  9  10:40 tmp->
/tmp
```

1.10 小结

本章介绍了一些有关文件安全的基本概念。如果这些命令能够使用得当而且使用得比较谨慎，应该不会有什么问题。手指轻轻一敲就有可能输入 chmod -R这样的命令，它将改变整个文件系统的权限，如果没有做备份的话，没有几年的时间恐怕是无法恢复了，所以在输入这些命令时，千万不要在手指上贴膏药！

是否使用设置了suid的脚本完全取决于你自己。如果使用的话，一定要确保能够监控它的使用，而且不要以根用户身份设置 suid。

China-pub.com 下载

第2章　使用find和xargs

有时可能需要在系统中查找具有某一特征的文件 (例如文件权限、文件属主、文件长度、文件类型等等)。这样做可能有很多原因。可能出于安全性的考虑，或是一般性的系统管理任务，或许只是为了找出一个不知保存在什么地方的文件。 Find是一个非常有效的工具，它可以遍历当前目录甚至于整个文件系统来查找某些文件或目录。

在本章中，我们介绍以下内容：

- find命令选项。
- 使用find命令不同选项的例子。
- 配合find使用xargs命令的例子。

由于find具有如此强大的功能，所以它的选项也很多，其中大部分选项都值得我们花时间来了解一下。即使系统中含有网络文件系统 (NFS)，find命令在该文件系统中同样有效，只要你具有相应的权限。

在运行一个非常消耗资源的find命令时，很多人都倾向于把它放在后台执行，因为遍历一个大的文件系统可能会花费很长的时间 (这里是指30G字节以上的文件系统)。

Find命令的一般形式为：

```
find pathname -options [-print -exec -ok]
```

让我们来看看该命令的参数：

pathname　find命令所查找的目录路径。例如用 .来表示当前目录，用 /来表示系统根目录。

-print　find命令将匹配的文件输出到标准输出。

-exec　find命令对匹配的文件执行该参数所给出的 shell命令。相应命令的形式为 'command' {} \;，注意{}和\；之间的空格。

-ok　和-exec的作用相同，只不过以一种更为安全的模式来执行该参数所给出的 shell命令，在执行每一个命令之前，都会给出提示，让用户来确定是否执行。

2.1　find命令选项

find命令有很多选项或表达式，每一个选项前面跟随一个横杠 -。让我们先来看一下该命令的主要选项，然后再给出一些例子。

-name　按照文件名查找文件。

-perm　按照文件权限来查找文件。

-prune　使用这一选项可以使find命令不在当前指定的目录中查找，如果同时使用了 -depth选项，那么 -prune选项将被find命令忽略。

-user　按照文件属主来查找文件。

-group　按照文件所属的组来查找文件。

-mtime -n +n　按照文件的更改时间来查找文件， -n表示文件更改时间距现在n天以内， +n表示文件更改时间距现在 n天以前。Find命令还有 -atime和-ctime选项，但它们都和 -mtime选项

China-pub.com 下载

相似，所以我们在这里只介绍-mtime选项。

-nogroup 查找无有效所属组的文件，即该文件所属的组在 /etc/groups中不存在。

-nouser 查找无有效属主的文件，即该文件的属主在 /etc/passwd中不存在。

-newer file1 ! file2 查找更改时间比文件file1新但比文件file2旧的文件。

-type 查找某一类型的文件，诸如：

b - 块设备文件。

d - 目录。

c - 字符设备文件。

p - 管道文件。

l - 符号链接文件。

f - 普通文件。

-size n[c] 查找文件长度为n块的文件，带有c时表示文件长度以字节计。

-depth 在查找文件时，首先查找当前目录中的文件，然后再在其子目录中查找。

-fstype 查找位于某一类型文件系统中的文件，这些文件系统类型通常可以在配置文件/etc/fstab中找到，该配置文件中包含了本系统中有关文件系统的信息。

-mount 在查找文件时不跨越文件系统mount点。

-follow 如果find命令遇到符号链接文件，就跟踪至链接所指向的文件。

-cpio 对匹配的文件使用cpio命令，将这些文件备份到磁带设备中。

2.1.1 使用name选项

文件名选项是find命令最常用的选项，要么单独使用该选项，要么和其他选项一起使用。可以使用某种文件名模式来匹配文件，记住要用引号将文件名模式引起来。

不管当前路径是什么，如果想要在自己的根目录 $HOME中查找文件名符合 *.txt的文件，使用~作为'pathname参数，波浪号~代表了你的$HOME目录。

```
$ find ~ -name "*.txt" -print
```

想要在当前目录及子目录中查找所有的‘ *.txt ’文件，可以用：

```
$ find . -name "*.txt" -print
```

想要的当前目录及子目录中查找文件名以一个大写字母开头的文件，可以用：

```
$ find . -name "[A-Z]*" -print
```

想要在/etc目录中查找文件名以host开头的文件，可以用：

```
$ find /etc -name "host*" -print
```

想要查找$HOME目录中的文件，可以用：

```
$ find ~ -name "*" -print或find . -print
```

要想让系统高负荷运行，就从根目录开始查找所有的文件。如果希望在系统管理员那里保留一个好印象的话，最好在这么做之前考虑清楚！

```
$ find / -name "*" -print
```

如果想在当前目录查找文件名以两个小写字母开头，跟着是两个数字，最后是 *.txt的文件，下面的命令就能够返回名为ax37.txt的文件：

China-pub.com 下载

```
$ find . -name "[a-z][a-z][0--9][0--9].txt" -print
```

2.1.2 使用perm选项

如果希望按照文件权限模式来查找文件的话，可以采用 -perm选项。你可能需要找到所有用户都具有执行权限的文件，或是希望查看某个用户目录下的文件权限类型。在使用这一选项的时候，最好使用八进制的权限表示法。

为了在当前目录下查找文件权限位为 755的文件，即文件属主可以读、写、执行，其他用户可以读、执行的文件，可以用：

```
$ find . -perm 755 -print
```

如果希望在当前目录下查找所有用户都可读、写、执行的文件（要小心这种情况），我们可以使用find命令的-perm选项。在八进制数字前面要加一个横杠 -。在下面的命令中 -perm代表按照文件权限查找，而‘ 007 ’和你在chmod命令的绝对模式中所采用的表示法完全相同。

```
$ find . -perm -007 -print
```

2.1.3 忽略某个目录

如果在查找文件时希望忽略某个目录，因为你知道那个目录中没有你所要查找的文件，那么可以使用-prune选项来指出需要忽略的目录。在使用 -prune选项时要当心，因为如果你同时使用了-depth选项，那么-prune选项就会被find命令忽略。

如果希望在/apps目录下查找文件，但不希望在 /apps/bin目录下查找，可以用：

```
$ find /apps -name "/apps/bin" -prune -o -print
```

2.1.4 使用user和nouser选项

如果希望按照文件属主查找文件，可以给出相应的用户名。例如，在 $HOME目录中查找文件属主为dave的文件，可以用：

```
$ find ~ -user dave -print
```

在/etc目录下查找文件属主为uucp的文件：

```
$ find /etc -user uucp -print
```

为了查找属主帐户已经被删除的文件，可以使用 -nouser选项。这样就能够找到那些属主在/etc/passwd文件中没有有效帐户的文件。在使用 -nouser选项时，不必给出用户名；find命令能够为你完成相应的工作。例如，希望在 /home目录下查找所有的这类文件，可以用：

```
$ find /home -nouser -print
```

2.1.5 使用group和nogroup选项

就像user和nouser选项一样，针对文件所属于的用户组， find命令也具有同样的选项，为了在/apps目录下查找属于accts用户组的文件，可以用：

```
$ find /apps -group accts -print
```

要查找没有有效所属用户组的所有文件，可以使用 nogroup选项。下面的find命令从文件系统的根目录处查找这样的文件

```
$ fine/-nogroup-print
```

China-pub.com
下载

2.1.6 按照更改时间查找文件

如果希望按照更改时间来查找文件，可以使用 mtime选项。如果系统突然没有可用空间了，很有可能某一个文件的长度在此期间增长迅速，这时就可以用 mtime选项来查找这样的文件。用减号 - 来限定更改时间在距今 n日以内的文件，而用加号 +来限定更改时间在距今 n日以前的文件。

希望在系统根目录下查找更改时间在 5日以内的文件，可以用：

```
$ find / -mtime -5 -print
```

为了在/var/adm目录下查找更改时间在 3日以前的文件，可以用：

```
$ find /var/adm -mtime +3 -print
```

2.1.7 查找比某个文件新或旧的文件

如果希望查找更改时间比某个文件新但比另一个文件旧的所有文件，可以使用 -newer选项。它的一般形式为：

```
newest_file_name ! oldest_file_name
```

其中，！是逻辑非符号。

这里有两个文件，它们的更改时间大约相差两天。

```
-rwxr-xr-x    1 root    root      92 Apr 18 11:18  age.awk
-rwxrwxr-x    1 root    root    1054 Apr 20 19:37  belts.awk
```

下面给出的find命令能够查找更改时间比文件 age.awk新但比文件belts.awk旧的文件：

```
$ find . -newer age.awk ! -newer belts.awk -exec ls -l {} \;
-rwxrwxr-x   1 root  root    62 Apr 18 11:32 ./who.awk
-rwxr-xr-x   1 root  root    49 Apr 18 12:05 ./group.awk
-rw-r--r--   1 root  root   201 Apr 20 19:30 ./grade2.txt
-rwxrwxr-x   1 root  root  1054 Apr 20 19:37 ./belts.awk
```

如果想使用find命令的这一选项来查找更改时间在两个小时以内的文件，除非有一个现成的文件其更改时间恰好在两个小时以前，否则就没有可用来比较更改时间的文件。为了解决这一问题，可以首先创建一个文件并将其日期和时间戳设置为所需要的时间。这可以用 touch命令来实现。

假设现在的时间是23:40，希望查找更改时间在两个小时以内的文件，可以首先创建这样一个文件：

```
$ touch -t 05042140 dstamp
$ ls -l dstamp
-rw-r--r--  1 dave     admin         0 May  4 21:40 dstamp
```

一个符合要求的文件已经被创建；这里我们假设今天是五月四日，而该文件的更改时间是21:40，比现在刚好早两个小时。

现在我们就可以使用 find命令的-newer选项在当前目录下查找所有更改时间在两个小时以内的文件：

```
$ find . -newer dstamp -print
```

2.1.8 使用type选项

UNIX或LINUX系统中有若干种不同的文件类型，这部分内容我们在前面的章节已经做了

China-pub.com 下载

介绍，这里就不再赘述。如果要在/etc目录下查找所有的目录，可以用：

```
$ find /etc -type d -print
```

为了在当前目录下查找除目录以外的所有类型的文件，可以用：

```
$ find . ! -type d -print
```

为了在/etc目录下查找所有的符号链接文件，可以用：

```
$ find /etc -type l -print
```

2.1.9 使用size选项

可以按照文件长度来查找文件，这里所指的文件长度既可以用块（block）来计量，也可以用字节来计量。以字节计量文件长度的表达形式为 Nc；以块计量文件长度只用数字表示即可。

就我个人而言，我总是使用以字节计的方式，在按照文件长度查找文件时，大多数人都喜欢使用这种以字节表示的文件长度，而不用块的数目来表示，除非是在查看文件系统的大小，因为这时使用块来计量更容易转换。

为了在当前目录下查找文件长度大于1M字节的文件，可以用：

```
$ find . -size +1000000c -print
```

为了在/home/apache目录下查找文件长度恰好为100字节的文件，可以用：

```
$ find /home/apache -size 100c -print
```

为了在当前目录下查找长度超过10块的文件（一块等于512字节），可以用：

```
$ find . -size +10 -print
```

2.1.10 使用depth选项

在使用find命令时，可能希望先匹配所有的文件，再在子目录中查找。使用 depth选项就可以使find命令这样做。这样做的一个原因就是，当在使用find命令向磁带上备份文件系统时，希望首先备份所有的文件，其次再备份子目录中的文件。

在下面的例子中，find命令从文件系统的根目录开始，查找一个名为 CON.FILE的文件。它将首先匹配所有的文件然后再进入子目录中查找。

```
$ find / -name "CON.FILE" -depth -print
```

2.1.11 使用mount选项

在当前的文件系统中查找文件（不进入其他文件系统），可以使用find命令的mount选项。在下面的例子中，我们从当前目录开始查找位于本文件系统中文件名以 XC结尾的文件：

```
$ find . -name "*.XC" -mount -print
```

2.1.12 使用cpio选项

cpio命令可以用来向磁带设备备份文件或从中恢复文件。可以使用 find命令在整个文件系统中（更多的情况下是在部分文件系统中）查找文件，然后用 cpio命令将其备份到磁带上。

如果希望使用cpio命令备份/etc、/home和/apps目录中的文件，可以使用下面所给出的命令，不过要记住你是在文件系统的根目录下：

```
$ cd /
$ find etc home apps -depth -print | cpio -ivcdC65536 -o  \
 /dev/rmt0
```

（在上面的例子中，第一行末尾的\告诉shell命令还未结束，忽略\后面的回车。）

在上面的例子中，应当注意到路径中缺少 /。这叫作相对路径。之所以使用相对路径，是因为在从磁带中恢复这些文件的时候，可以选择恢复文件的路径。例如，可以将这些文件先恢复到另外一个目录中，对它们进行某些操作后，再恢复到原始目录中。如果在备份时使用了绝对路径，例如 /etc，那么在恢复时，就只能恢复到 /etc目录中去，别无其他选择。在上面的例子中，我告诉 find命令首先进入/etc目录，然后是/home和/apps目录，先匹配这些目录下的文件，然后再匹配其子目录中的文件，所有这些结果将通过管道传递给 cpio命令进行备份。

顺便说一下，在上面的例子中cpio命令使用了C65536选项，我本可以使用B选项，不过这样每块的大小只有 512字节，而使用了 C65536选项后，块的大小变成了 64K字节（65536/1024）。

2.1.13 使用exec或ok来执行shell命令

当匹配到一些文件以后，可能希望对其进行某些操作，这时就可以使用 -exec选项。一旦find命令匹配到了相应的文件，就可以用 -exec选项中的命令对其进行操作（在有些操作系统中只允许-exec选项执行诸如ls或ls -l这样的命令）。大多数用户使用这一选项是为了查找旧文件并删除它们。这里我强烈地建议你在真正执行 rm命令删除文件之前，最好先用 ls命令看一下，确认它们是所要删除的文件。

exec选项后面跟随着所要执行的命令，然后是一对儿 {}，一个空格和一个\，最后是一个分号。

为了使用exec选项，必须要同时使用 print选项。如果验证一下 find命令，会发现该命令只输出从当前路径起的相对路径及文件名。

为了用ls -l命令列出所匹配到的文件，可以把 ls -l命令放在find命令的-exec选项中，例如：

```
$ find . -type f  -exec ls -l {} \;
-rwxr-xr-x  10 root  wheel        1222 Jan  4  1993./sbin/C80
-rwxr-xr-x  10 root  wheel        1222 Jan  4  1993./sbin/Normal
-rwxr-xr-x  10 root  wheel        1222 Jan  4  1993./sbin/Revvid
```

上面的例子中， find命令匹配到了当前目录下的所有普通文件，并在 -exec选项中使用ls -l命令将它们列出。

为了在/logs目录中查找更改时间在5日以前的文件并删除它们，可以用：

```
$ find logs -type f -mtime +5 -exec rm {} \;
```

记住，在shell中用任何方式删除文件之前，应当先查看相应的文件，一定要小心！

当使用诸如mv或rm命令时，可以使用-exec选项的安全模式。它将在对每个匹配到的文件进行操作之前提示你。在下面的例子中， find命令在当前目录中查找所有文件名以.LOG结尾、更改时间在5日以上的文件，并删除它们，只不过在删除之前先给出提示。

```
$ find . -name "*.LOG" -mtime +5 -ok rm {} \;
< rm ... ./nets.LOG > ? y
```

按y键删除文件，按n键不删除。

任何形式的命令都可以在-exec选项中使用。在下面的例子中我们使用 grep命令。find命令

China-pub.com 下载

首先匹配所有文件名为“ passwd* ”的文件，例如passwd、passwd.old、passwd.bak，然后执行grep命令看看在这些文件中是否存在一个rounder用户。

```
$ find /etc -name "passwd*" -exec grep "rounder" {} \;
rounder:JL9TtUqk8EHwc:500:500::/home/apps/nets/rounder:/bin/sh
```

2.1.14　find命令的例子

我们已经介绍了find命令的基本选项，下面给出find命令的一些其他的例子。

为了匹配$HOME目录下的所有文件，下面两种方法都可以使用：

```
$ find $HOME -print
$ find ~ -print
```

为了在当前目录中查找suid置位，文件属主具有读、写、执行权限，并且文件所属组的用户和其他用户具有读和执行的权限的文件，可以用：

```
$ find . -type f -perm 4755 -print
```

为了查找系统中所有文件长度为0的普通文件，并列出它们的完整路径，可以用：

```
$ find / -type f -size 0 -exec ls -l {} \;
```

为了查找/var/logs目录中更改时间在7日以前的普通文件，并删除它们，可以用：

```
$ find /var/logs -type f -mtime +7 -exec rm {} \;
```

为了查找系统中所有属于audit组的文件，可以用：

```
$find /-name -group audit -print
```

我们的一个审计系统每天创建一个审计日志文件。日志文件名的最后含有数字，这样我们一眼就可以看出哪个文件是最新的，哪个是最旧的。　Admin.log文件编上了序号：admin.log.001、admin.log.002等等。下面的find命令将删除/logs目录中访问时间在7日以前、含有数字后缀的admin.log文件。该命令只检查三位数字，所以相应日志文件的后缀不要超过999。

```
$ find /logs -name 'admin.log[0-9][0-9][0-9]' -atime +7 -exec rm {} \;
```

为了查找当前文件系统中的所有目录并排序，可以用：

```
$ find . -type d -print -local -mount |sort
```

为了查找系统中所有的rmt磁带设备，可以用：

```
$ find /dev/rmt -print
```

2.2　xargs

在使用find命令的-exec选项处理匹配到的文件时，find命令将所有匹配到的文件一起传递给exec执行。不幸的是，有些系统对能够传递给exec的命令长度有限制，这样在find命令运行几分钟之后，就会出现溢出错误。错误信息通常是“参数列太长”或“参数列溢出”。这就是xargs命令的用处所在，特别是与find命令一起使用。Find命令把匹配到的文件传递给xargs命令，而xargs命令每次只获取一部分文件而不是全部，不像-exec选项那样。这样它可以先处理最先获取的一部分文件，然后是下一批，并如此继续下去。在有些系统中，使用-exec选项会为处理每一个匹配到的文件而发起一个相应的进程，并非将匹配到的文件全部作为参数一次执行；这样在有些情况下就会出现进程过多，系统性能下降的问题，因而效率不高；而使用

China-pub.com 下载

xargs命令则只有一个进程。另外，在使用 xargs命令时，究竟是一次获取所有的参数，还是分批取得参数，以及每一次获取参数的数目都会根据该命令的选项及系统内核中相应的可调参数来确定。

让我们来看看xargs命令是如何同find命令一起使用的，并给出一些例子。

下面的例子查找系统中的每一个普通文件，然后使用 xargs命令来测试它们分别属于哪类文件：

```
$ find / -type f  -print | xargs file
/etc/protocols: English text
/etc/securetty: ASCII text
..
```

下面的例子在整个系统中查找内存信息转储文件 (core dump)，然后把结果保存到/tmp/core.log 文件中：

```
$ find . -name "core" -print | xargs echo "" >/tmp/core.log
```

下面的例子在/apps/audit目录下查找所有用户具有读、写和执行权限的文件，并收回相应的写权限：

```
$ find /apps/audit -perm -7 -print | xargs chmod o-w
```

在下面的例子中，我们用grep命令在所有的普通文件中搜索device这个词：

```
$ find / -type f -print | xargs grep "device"
```

在下面的例子中，我们用grep命令在当前目录下的所有普通文件中搜索 DBO这个词：

```
$ find . -name *\-type f -print | xargs grep "DBO"
```

注意，在上面的例子中，\用来取消find命令中的*在shell中的特殊含义。

2.3　小结

find命令是一个非常优秀的工具，它可以按照用户指定的准则来匹配文件。使用 exec和xargs可以使用户对所匹配到的文件执行几乎所有的命令。

China-pub.com
下载

第3章 后台执行命令

当你在终端或控制台工作时，可能不希望由于运行一个作业而占住了屏幕，因为可能还有更重要的事情要做，比如阅读电子邮件。对于密集访问磁盘的进程，你可能希望它能够在每天的非负荷高峰时间段运行。为了使这些进程能够在后台运行，也就是说不在终端屏幕上运行，有几种选择方法可供使用。

在本章中我们将讨论：

• 设置crontab文件，并用它来提交作业。
• 使用at命令来提交作业。
• 在后台提交作业。
• 使用nohup命令提交作业。

名词解释：

cron 系统调度进程。可以使用它在每天的非高峰负荷时间段运行作业，或在一周或一月中的不同时段运行。

At at命令。使用它在一个特定的时间运行一些特殊的作业，或在晚一些的非负荷高峰时间段或高峰负荷时间段运行。

& 使用它在后台运行一个占用时间不长的进程。

Nohup 使用它在后台运行一个命令，即使在用户退出时也不受影响。

3.1 cron和crontab

cron是系统主要的调度进程，可以在无需人工干预的情况下运行作业。有一个叫做crontab的命令允许用户提交、编辑或删除相应的作业。每一个用户都可以有一个 crontab文件来保存调度信息。可以使用它运行任意一个 shell脚本或某个命令，每小时运行一次，或一周三次，这完全取决于你。每一个用户都可以有自己的 crontab文件，但在一个较大的系统中，系统管理员一般会禁止这些文件，而只在整个系统保留一个这样的文件。系统管理员是通过cron.deny和cron.allow这两个文件来禁止或允许用户拥有自己的 crontab文件。

3.1.1 crontab的域

为了能够在特定的时间运行作业，需要了解 crontab文件每个条目中各个域的意义和格式。下面就是这些域：

第1列　　分钟1～59
第2列　　小时1～23（0表示子夜）
第3列　　日1～31
第4列　　月1～12
第5列　　星期0～6（0表示星期天）
第6列　　要运行的命令

下面是crontab的格式：

分<>时<>日<>月<>星期<>要运行的命令

其中<>表示空格。

Crontab文件的一个条目是从左边读起的，第一列是分，最后一列是要运行的命令，它位于星期的后面。

在这些域中，可以用横杠 -来表示一个时间范围，例如你希望星期一至星期五运行某个作业，那么可以在星期域使用 1-5来表示。还可以在这些域中使用逗号“，”，例如你希望星期一和星期四运行某个作业，只需要使用 1,4来表示。可以用星号 *来表示连续的时间段。如果你对某个表示时间的域没有特别的限定，也应该在该域填入 *。该文件的每一个条目必须含有 5个时间域，而且每个域之间要用空格分隔。该文件中所有的注释行要在行首用 #来表示。

3.1.2 crontab条目举例

这里有crontab文件条目的一些例子：

```
30 21* * * /apps/bin/cleanup.sh
```

上面的例子表示每晚的21:30运行/apps/bin目录下的cleanup.sh。

```
45 4 1,10,22 * * /apps/bin/backup.sh
```

上面的例子表示每月1、10、22日的4:45运行/apps/bin目录下的backup.sh。

```
10 1 * * 6,0 /bin/find -name "core" -exec rm {} \;
```

上面的例子表示每周六、周日的1:10运行一个find命令。

```
0,30 18-23 * * * /apps/bin/dbcheck.sh
```

上面的例子表示在每天18:00至23:00之间每隔30分钟运行/apps/bin目录下的dbcheck.sh。

```
0 23 * * 6 /apps/bin/qtrend.sh
```

上面的例子表示每星期六的11:00pm运行/apps/bin目录下的qtrend.sh。

你可能已经注意到上面的例子中，每个命令都给出了绝对路径。当使用 crontab运行shell脚本时，要由用户来给出脚本的绝对路径，设置相应的环境变量。记住，既然是用户向 cron提交了这些作业，就要向cron提供所需的全部环境。不要假定cron知道所需要的特殊环境，它其实并不知道。所以你要保证在 shell脚本中提供所有必要的路径和环境变量，除了一些自动设置的全局变量。

如果cron不能运行相应的脚本，用户将会收到一个邮件说明其中的原因。

3.1.3 crontab命令选项

crontab命令的一般形式为：

```
Crontab [-u user] -e -l -r
```

其中：

-u 用户名。

-e 编辑crontab文件。

-l 列出crontab文件中的内容。

-r 删除crontab文件。

如果使用自己的名字登录，就不用使用 -u选项，因为在执行 crontab命令时，该命令能够

China-pub.com 下载

知道当前的用户。

3.1.4 创建一个新的crontab文件

在考虑向cron进程提交一个crontab文件之前，首先要做的一件事情就是设置环境变量EDITOR。cron进程根据它来确定使用哪个编辑器编辑crontab文件。99%的UNIX和LINUX用户都使用vi，如果你也是这样，那么你就编辑$HOME目录下的.profile文件，在其中加入这样一行：

```
EDITOR=vi; export EDITOR
```

然后保存并退出。

不妨创建一个名为<user>cron的文件，其中<user>是用户名，例如，davecron。在该文件中加入如下的内容。

```
# (put your own initials here) echo the date to the console every
# 15 minutes between 6pm and 6am
0,15,30,45 18-06  *  *  *  /bin/echo 'date' > /dev/console
```

保存并退出。确信前面5个域用空格分隔。

在上面的例子中，系统将每隔15分钟向控制台输出一次当前时间。如果系统崩溃或挂起，从最后所显示的时间就可以一眼看出系统是什么时间停止工作的。在有些系统中，用tty1来表示控制台，可以根据实际情况对上面的例子进行相应的修改。

为了提交你刚刚创建的crontab文件，可以把这个新创建的文件作为cron命令的参数：

```
$ crontab davecron
```

现在该文件已经提交给cron进程，它将每隔15分钟运行一次。

同时，新创建文件的一个副本已经被放在/var/spool/cron目录中，文件名就是用户名（即，dave）。

3.1.5 列出crontab文件

为了列出crontab文件，可以用：

```
$ crontab -l
# (crondave installed on Tue May  4 13:07:43 1999)
# DT: echo the date to the console every 30 minutes
 0,15,30,45 18-06  *  *  *  /bin/echo 'date' > /dev/tty1
```

你将会看到和上面类似的内容。可以使用这种方法在$HOME目录中对crontab文件做一备份：

```
$ crontab -l > $HOME/mycron
```

这样，一旦不小心误删了crontab文件，可以用上一节所讲述的方法迅速恢复。

3.1.6 编辑crontab文件

如果希望添加、删除或编辑crontab文件中的条目，而EDITOR环境变量又设置为vi，那么就可以用vi来编辑crontab文件，相应的命令为：

```
$ crontab -e
```

可以像使用vi编辑其他任何文件那样修改crontab文件并退出。如果修改了某些条目或添

China-pub.com
下载

加了新的条目，那么在保存该文件时， cron会对其进行必要的完整性检查。如果其中的某个域出现了超出允许范围的值，它会提示你。

我们在编辑crontab文件时，没准会加入新的条目。例如，加入下面的一条：

```
# DT: delete core files, at 3.30am on 1,7,14,21,26 days of each month
30 3 1,7,14,21,26 * * /bin/find -name "core" -exec rm {} \;
```

现在保存并退出。最好在crontab文件的每一个条目之上加入一条注释，这样就可以知道它的功能、运行时间，更为重要的是，知道这是哪位用户的作业。

现在让我们使用前面讲过的crontab -l命令列出它的全部信息：

```
$ crontab -l
# (crondave installed on Tue May  4 13:07:43 1999)
# DT: echo the date to the console every 30 minutes
0,15,30,45 18-06 * * * /bin/echo 'date' > /dev/tty1
# DT: delete core files, at 3.30am on 1,7,14,21,26 days of each
# month
31 3 1,7,14,21,26 * * /bin/find -name "core" -exec rm {} \;
```

3.1.7 删除crontab文件

为了删除crontab文件，可以用：

```
$ crontab -r
```

3.1.8 恢复丢失的crontab文件

如果不小心误删了crontab文件，假设你在自己的$HOME目录下还有一个备份，那么可以将其拷贝到/var/spool/cron/<username>，其中<username>是用户名。如果由于权限问题无法完成拷贝，可以用：

```
$ crontab <filename>
```

其中，<filename>是你在$HOME目录中副本的文件名。

我建议你在自己的$HOME目录中保存一个该文件的副本。我就有过类似的经历，有数次误删了crontab文件（因为r键紧挨在e键的右边…）。这就是为什么有些系统文档建议不要直接编辑crontab文件，而是编辑该文件的一个副本，然后重新提交新的文件。

有些crontab的变体有些怪异，所以在使用crontab命令时要格外小心。如果遗漏了任何选项，crontab可能会打开一个空文件，或者看起来像是个空文件。这时敲delete键退出，不要按<Ctrl-D>，否则你将丢失crontab文件。

3.2 at命令

at命令允许用户向cron守护进程提交作业，使其在稍后的时间运行。这里稍后的时间可能是指10min以后，也可能是指几天以后。如果你希望在一个月或更长的时间以后运行，最好还是使用crontab文件。

一旦一个作业被提交，at命令将会保留所有当前的环境变量，包括路径，不象crontab，只提供缺省的环境。该作业的所有输出都将以电子邮件的形式发送给用户，除非你对其输出进行了重定向，绝大多数情况下是重定向到某个文件中。

和crontab一样，根用户可以通过/etc目录下的at.allow和at.deny文件来控制哪些用户可以

China-pub.com 下载

使用at命令，哪些用户不行。不过一般来说，对 at命令的使用不如对 crontab的使用限制那么严格。

at命令的基本形式为：

```
at [-f script] [-m -l -r] [time] [date]
```

其中，

-f script 是所要提交的脚本或命令。

-l 列出当前所有等待运行的作业。 atq命令具有相同的作用。

-r 清除作业。为了清除某个作业，还要提供相应的作业标识（ ID）；有些UNIX变体只接受atrm作为清除命令。

-m 作业完成后给用户发邮件。

time at命令的时间格式非常灵活；可以是 H、HH.HHMM、HH:MM或H:M，其中H和M分别是小时和分钟。还可以使用 a.m.或p.m.。

date 日期格式可以是月份数或日期数，而且 at命令还能够识别诸如today、tomorrow这样的词。

现在就让我们来看看如何提交作业。

3.2.1 使用at命令提交命令或脚本

使用at命令提交作业有几种不同的形式，可以通过命令行方式，也可以使用 at命令提示符。一般来说在提交若干行的系统命令时，我使用 at命令提示符方式，而在提交 shell脚本时，使用命令行方式。

如果你想提交若干行的命令，可以在 at命令后面跟上日期/时间并回车。然后就进入了 at命令提示符，这时只需逐条输入相应的命令，然后按‘ <CTRL-D> ’退出。下面给出一个例子：

```
$ at 21:10
at> find / -name "passwd" -print
at> <EOT>
warning: commands will be executed using /bin/sh
job 1 at 1999-05-05 21:10
```

其中， <EOT>就是<CTRL-D>。在21:10系统将执行一个简单的 find命令。你应当已经注意到，我所提交的作业被分配了一个唯一标识 job 1。该命令在完成以后会将全部结果以邮件的形式发送给我。

下面就是我从这个邮件中截取的一部分：

```
Subject: Output from your job      1
/etc/passwd
/etc/pam.d/passwd
/etc/uucp/passwd
/tmp/passwd
/root/passwd
/usr/bin/passwd
/usr/doc/uucp-1.06.1/sample/passwd
```

下面这些日期/时间格式都是at命令可以接受的：

```
At 6.45am May12
At 11.10pm
At now + 1 hour
```

China-pub.com
下载

At 9am tomorrow
At 15:00 May 24
At now + 10 minutes – this time specification is my own favourite.

如果希望向at命令提交一个shell脚本，使用其命令行方式即可。在提交脚本时使用 -f选项。

```
$ at 3.00pm tomorrow -f /apps/bin/db_table.sh
warning: commands will be executed using /bin/sh
job 8 at 1999-05-06 15:00
```

在上面的例子中，一个叫做db_table.sh的脚本将在明天下午3:00运行。

还可以使用echo命令向at命令提交作业：

```
$ echo find /etc -name "passwd" -print | at now +1 minute
```

3.2.2 列出所提交的作业

一个作业被提交后，可以使用at -l命令来列出所有的作业：

```
$ at -l
2    1999-05-05 23:00 a
3    1999-05-06 06:00 a
4    1999-05-21 11:20 a
1    1999-05-06 09:00 a
```

其中，第一行是作业标识，后面是作业运行的日期 /时间。最后一列a代表at。还可以使用atq命令来完成同样的功能，它是 at命令的一个链接。当提交一个作业后，它就被拷贝到/var/spool/at目录中，准备在要求的时间运行。

```
$ pwd
/var/spool/at
$ ls
a0000200eb7ae4  a0000400ebd228  a0000800eb7ea4  spool
a0000300eb7c88  a0000500eb7d3c  a0000900eb7aaa
```

3.2.3 清除一个作业

清除作业的命令格式为：

```
atrm [job no]或at -r [job no]
```

要清除某个作业，首先要执行at -l命令，以获取相应的作业标识，然后对该作业标识使用at -r命令，清除该作业。

```
$ at -l
2    1999-05-05 23:00 a
3    1999-05-06 06:00 a
4    1999-05-21 11:20 a

$ atrm job 3
$ at -l
2    1999-05-05 23:00 a
4    1999-05-21 11:20 a
```

有些系统使用at-r [job no]命令清除作业。

3.3 &命令

当在前台运行某个作业时，终端被该作业占据；而在后台运行作业时，它不会占据终端。

China-pub.com 下载

可以使用&命令把作业放到后台执行。

该命令的一般形式为：

```
命令 &
```

为什么要在后台执行命令？因为当在后台执行命令时，可以继续使用你的终端做其他事情。适合在后台运行的命令有find、费时的打印作业、费时的排序及一些shell脚本。在后台运行作业时要当心：需要用户交互的命令不要放在后台执行，因为这样你的机器就会在那里傻等。

不过，作业在后台运行一样会将结果输出到屏幕上，干扰你的工作。如果放在后台运行的作业会产生大量的输出，最好使用下面的方法把它的输出重定向到某个文件中：

```
command >out.file 2>&1 &
```

在上面的例子中，所有的标准输出和错误输出都将被重定向到一个叫做 out.file 的文件中。当你成功地提交进程以后，就会显示出一个进程号，可以用它来监控该进程，或杀死它。

3.3.1 向后台提交命令

现在我们运行一个find命令，查找名为“ srm.conf ”的文件，并把所有标准输出和错误输出重定向到一个叫作find.dt的文件中：

```
$ find /etc -name "srm.conf" -print >find.dt 2>&1 &
[1] 27015
```

在上面的例子中，在我们成功提交该命令之后，系统给出了它的进程号 27015。

当该作业完成时，按任意键（一般是回车键）就会出现一个提示：

```
[1]+  Done          find /etc "srm.conf" -print
```

这里还有另外一个例子，有一个叫做 ps1的脚本，它能够截断和清除所有的日志文件，我把它放到后台去执行：

```
$ ps1 &
[2] 28535
```

3.3.2 用ps命令查看进程

当一个命令在后台执行的时候，可以用提交命令时所得到的进程号来监控它的运行。在前面的例子中，我们可以按照提交 ps1时得到的进程号，用ps命令和grep命令列出这个进程：

```
$ ps x|grep 28305
28305  p1 S    0:00 sh /root/ps1
28350  p1 S    0:00 grep 28305
```

如果系统不支持ps x命令，可以用：

```
$ ps -ef |grep 28305
root 28305 21808    0 10:24:39  pts/2   0:00 sh ps1
root 21356 21808    1 10:24:46  pts/2   0:00 grep 28305
```

记住，在用ps命令列出进程时，它无法确定该进程是运行在前台还是后台。

3.3.3 杀死后台进程

如果想杀死后台进程可以使用 kill命令。当一个进程被放到后台运行时， shell会给出一个

进程号，我们可以根据这个进程号，用kill命令杀死该进程。该命令的基本形式为：

```
kill -signal [process_number]
```

现在暂且不要考虑其中的各种不同信号；我们会在后面的章节对这一问题进行介绍。

在杀进程的时候，执行下面的命令(你的进程号可能会不同)并按回车键。系统将会给出相应的信息告诉用户进程已经被杀死。

```
$ kill 28305
[1]+  Terminated              ps1
```

如果系统没有给出任何信息，告诉你进程已经被杀死，那么不妨等一会儿，也许系统正在杀该进程，如果还没有回应，就再执行另外一个kill命令，这次带上一个信号选项：

```
$ kill -9 28305
[1] + Killed                   ps1 &
```

如果用上述方法提交了一个后台进程，那么在退出时该进程将会被终止。为了使后台进程能够在退出后继续运行，可以使用nohup命令，下面我们就介绍这一命令。

3.4 nohup命令

如果你正在运行一个进程，而且你觉得在退出帐户时该进程还不会结束，那么可以使用nohup命令。该命令可以在你退出帐户之后继续运行相应的进程。Nohup就是不挂起的意思(no hang up)。

该命令的一般形式为：

```
nohup command &
```

3.4.1 使用nohup命令提交作业

如果使用nohup命令提交作业，那么在缺省情况下该作业的所有输出都被重定向到一个名为nohup.out的文件中，除非另外指定了输出文件：

```
nohup command > myout.file 2>&1
```

在上面的例子中，输出被重定向到myout.file文件中。

让我们来看一个例子，验证一下在退出帐户后相应的作业是否能够继续运行。我们先提交一个名为ps1的日志清除进程：

```
$ nohup ps1 &
[1] 179
$ nohup: appending output to 'nohup.out'
```

现在退出该shell，再重新登录，然后执行下面的命令：

```
$ ps x |grep ps1
  179  ?   S N   0:01 sh /root/ps1
  506  p2 S      0:00 grep ps1
```

我们看到，该脚本还在运行。如果系统不支持ps x命令，使用ps -ef|grep ps1命令。

3.4.2 一次提交几个作业

如果希望一次提交几个命令，最好能够把它们写入到一个shell脚本文件中，并用nohup命令来执行它。例如，下面的所有命令都用管道符号连接在一起；我们可以把这些命令存入一

China-pub.com 下载

个文件，并使该文件可执行。

```
cat /home/accounts/qtr_0499 | /apps/bin/trials.awk |sort|lp
```

```
$ cat > quarterend
cat /home/accounts/qtr_0499 | /apps/bin/trials.awk |sort|lp
<CTRL-D>
```

现在让它可执行：

```
$ chmod 744 quarterend
```

我们还将该脚本的所有输出都重定向到一个名为 qtr.out的文件中。

```
$ nohup ./quarterend > qtr.out 2>&1 &
[5] 182
```

3.5 小结

本章中所讨论的工具主要是有关后台运行作业的。有时我们必须要对大文件进行大量更改，或执行一些复杂的查找，这些工作最好能够在系统负荷较低时执行。

创建一个定时清理日志文件或完成其他特殊工作的脚本，这样只要提交一次，就可以每天晚上运行，而且无需你干预，只要看看相应的脚本日志就可以了。 Cron和其他工具可以使系统管理任务变得更轻松。

China-pub.com
下载

第4章　文件名置换

当你在使用命令行时，有很多时间都用来查找你所需要的文件。 Shell提供了一套完整的字符串模式匹配规则，或者称之为元字符，这样你就可以按照所要求的模式来匹配文件。还可以使用字符类型来匹配文件名。在命令行方式下，使用元字符更为快捷，所以在本章我们只介绍这部分内容。

在本章我们将讨论：

• 匹配文件名中的任何字符串。
• 匹配文件名中的单个字符。
• 匹配文件名中的字母或数字字符。

下面就是这些特殊字符：

*　匹配文件名中的任何字符串，包括空字符串。

?　匹配文件名中的任何单个字符。

[...]　匹配[ ]中所包含的任何字符。

[!...]　匹配[ ]中非感叹号！之后的字符。

当shell遇到上述字符时，就会把它们当作特殊字符，而不是文件名中的普通字符，这样用户就可以用它们来匹配相应的文件名。

4.1　使用*

使用星号*可以匹配文件名中的任何字符串。在下面的例子中，我们给出文件名模式 app*，它的意思是文件名以app开头，后面可以跟随任何字符串，包括空字符串：

```
$ ls app*
appdva          app_tapes
appdva_SLA
```

*也可以用在文件名模式的开头，在下面的例子中， *.doc匹配所有以.doc结尾的 文件名：

```
$ ls *.doc
accounts.doc    qtr_end.doc
```

还可以用在文件名的当中，在下面的例子中， cl.sed用于匹配所有以cl开头、后面跟任何字符串、最后以.sed结尾的文件名：

```
$ ls cl*.sed
cleanmeup.sed          cleanlogs.sed
cleanmessages.sed
```

在使用cd命令切换路径时，使用星号还可以省去输入整个路径名的麻烦，下面给出一个这样的例子：

```
$ pwd
/etc
$ ls -l |grep ^d
...
```

China-pub.com 下载

```
...
drwxr-xr-x   2 root     root      1024 Jan 26 14:41 cron.daily
drwxr-xr-x   2 root     root      1024 Jun 27  1998 cron.hourly
drwxr-xr-x   2 root     root      1024 Jun 27  1998 cron.monthly
drwxr-xr-x   2 root     root      1024 Jan 26 14:37 cron.weekly
...
$ cd cron.w*
$ pwd
/etc/cron.weekly
```

4.2 使用？

使用可以匹配文件名中的任何单个字符。在下面的例子中，我们列出文件名以任意两个字符开头，接着是R，后面跟任何字符的文件：

```
$ ls ??R*
BAREAD
```

在下面的例子中，我们列出文件名以 conf开头、中间是任意两个字符、最后以 .log结尾的文件：

```
$ ls conf.??.log
conf12.log conf.23.log
conf25.log
```

在下面的例子中， f??*s匹配所有以f开头、中间是任意两个字符、后面跟随任意字符串、并以s结尾的文件名：

```
$ ls f??*s
ftpaccess        ftpconversions
ftpgroups        ftphosts
ftpusers
```

4.3 使用[...]和[!...]

使用[...]可以用来匹配方括号[ ]中的任何字符。在这一方法中，还可以使用一个横杠 -来连接两个字母或数字，以此来表示一个范围。在下面的例子中，列出了以 i或o开头的文件名：

```
$ ls [io]*
inetd.conf  initrunlvl
inputrc     issue
info-dir    inittab
ioctl.save  issue.net
```

为了匹配所有以log.开头、后面跟随一个数字、然后可以是任意字符串的文件名，可以用log.[0-9]*，其中[0-9]表示任意单个数字，星号*代表了其他字符：

```
$ ls log.[0-9]*
log.0323  log.0324
log.0325  log.0326
```

下面的例子和刚才的有所不同，使用 [!0-9]*来表示非数字开头的字符串，其中 !是非的意思：

```
$ ls log.[!0-9]*
log.sybase
```

下面的例子中，列出了所有以LPS开头、中间可以是任何两个字符，最后以1结尾的文件名：

China-pub.com
下载


```
$ ls LPS??[1]
LPSI91  LPSO91
```

下面的例子中，列出了所有以LPS开头、中间可以是任何两个字符，后面跟随一个非数字字符、然后是任意字符串的文件名：

```
$ ls LPS??[!0-9]*
LPSOSI       LPSILP
LPSOPS       LPSPOPQTR
```

为了列出所有以大写字母开头的文件名，可以用：

```
$ ls [A-Z]*
```

为了列出所有以小写字母开头的文件名，可以用：

```
$ ls [a-z]*
```

为了列出所有以数字开头的文件名，可以用：

```
$ ls [0-9]*
```

为了列出所有以 . 开头的文件名（隐含文件，例如 .profile、.rhosts、.history等等），可以用：

```
$ ls .*
```

4.4 小结

使用元字符可以大大减少你在查找文件名上的工作量。这是一种非常有效的模式匹配方法，在后面的章节中，我们还将在讨论正则表达式的时候对文本处理中所涉及到的元字符进行更为详尽的讨论。

第5章　shell输入与输出

在shell脚本中，可以用几种不同的方式读入数据：可以使用标准输入——缺省为键盘，或者指定一个文件作为输入。对于输出也是一样：如果不指定某个文件作为输出，标准输出总是和终端屏幕相关联。如果所使用命令出现了什么错误，它也会缺省输出到屏幕上，如果不想把这些信息输出到屏幕上，也可以把这些信息指定到一个文件中。

大多数使用标准输入的命令都指定一个文件作为标准输入。如果能够从一个文件中读取数据，何必要费时费力地从键盘输入呢？

本章我们将讨论以下内容：

• 使用标准输入、标准输出及标准错误。

• 重定向标准输入和标准输出。

本章全面讨论了shell对数据和信息的标准输入、标准输出，对重定向也做了一定的介绍。

5.1　echo

使用echo命令可以显示文本行或变量，或者把字符串输入到文件。它的一般形式为：

```
echo string
```

echo命令有很多功能，其中最常用的是下面几个：

\c　不换行。

\f　进纸。

\t　跳格。

\n　换行。

如果希望提示符出现在输出的字符串之后，可以用：

```
$ echo  "What is your name :\c"
$ read name
```

上面的命令将会有如下的显示：

```
What is your name :□
```

其中“□”是光标。

如果想在输出字符之后，让光标移到下一行，可以用：

```
$ echo "The red pen ran out of ink"
```

还可以用echo命令输出转义符以及变量。在下面的例子中，你可以让终端铃响一声，显示出$HOME目录，并且可以让系统执行tty命令(注意，该命令用键盘左上角的符号，法语中的抑音符引起来，不是单引号，)。

```
$ echo "\007your home directory is $HOME, you are connected on `tty`"

your home directory is /home/dave, you are connected on /dev/ttyp1
```

如果是LINUX系统，那么......

必须使用-n选项来禁止echo命令输出后换行：

(续)

```
$ echo -n "What is your name :"
```

必须使用-e选项才能使转义符生效：

```
$ echo -e "\007your home directory is $HOME, you are connected on
`tty`"

your home directory is /home/dave, you are connected on /dev/ttyp1
```

如果希望在echo命令输出之后附加换行，可以使用\n选项：

```
$ pg echod
#!/bin/sh
echo  "this echo's 3 new lines\n\n\n"
echo "OK"
```

运行时会出现如下输出：

```
$ echod
this echo's 3 blank lines



OK
```

还可以在echo语句中使用跳格符，记住别忘了加反斜杠\：

```
$ echo "here is a tab\there are two tabs\t\tok"
here is a tab   here are two tabs               ok
```

如果是LINUX系统，那么...

别忘了使用- e选项才能使转义符生效：

```
$ echo -e "here is a tab\there are two tabs\t\tok"
here is a tab   here are two tabs               ok
```

如果想把一个字符串输出到文件中，使用重定向符号 >。在下面的例子中一个字符串被重定向到一个名为myfile的文件中：

```
$ echo "The log files have all been done"> myfile
```

或者可以追加到一个文件的末尾，这意味着不覆盖原有的内容：

```
$ echo "$LOGNAME carried them out at `date`">>myfile
```

现在让我们看一下myfile文件中的内容：

```
$ pg myfile
 The log files have all been done
 root carried them out at Sat May 22 18:25:06 GMT 1999
```

初涉shell的用户常常会遇到的一个问题就是如何把双引号包含到 echo命令的字符串中。引号是一个特殊字符，所以必须要使用反斜杠 \来使shell忽略它的特殊含义。假设你希望使用echo命令输出这样的字符串：“ /dev/rmt0 ”，那么我们只要在引号前面加上反斜杠\即可：

```
$ echo "\"/dev/rmt0"\"
"/dev/rmt0"
```

5.2 read

可以使用read语句从键盘或文件的某一行文本中读入信息，并将其赋给一个变量。如果只

China-pub.com 下载

指定了一个变量，那么read将会把所有的输入赋给该变量，直至遇到第一个文件结束符或回车。

它的一般形式为：

```
read varible1 varible2 ...
```

在下面的例子中，只指定了一个变量，它将被赋予直至回车之前的所有内容：

```
$ read name
Hello I am superman
$ echo $name
Hello I am superman
```

在下面的例子中，我们给出了两个变量，它们分别被赋予名字和姓氏。 shell将用空格作为变量之间的分隔符：

```
$ read name surname
John Doe
$ echo $name $surname
John Doe
```

如果输入文本域过长， Shell 将所有的超长部分赋予最后一个变量。下面的例子，假定要读取变量名字和姓，但这次输入三个名字；结果如下；

```
$ read name surname
John Lemon Doe
$ echo $name
John
$ echo $surname
Lemon Doe
```

在上面的例子中，如果我们输入字符串 John Lemon Doe，那么第一个单词将被赋给第一个变量，而由于变量数少于单词数，字符串后面的部分将被全部赋给第二个变量。

在编写shell脚本的时候，如果担心用户会对此感到迷惑，可以采用每一个 read语句只给一个变量赋值的办法：

```
$ pg var_test
#!/bin/sh
# var_test
echo "First Name :\c"
read name
echo "Middle Name :\c"
read middle
echo "Last name :\c"
read surname
```

用户在运行上面这个脚本的时候，就能够知道哪些信息赋给了哪个变量。

```
$ var_test
First Name : John
Middle Name : Lemon
Surname : Doe
```

如果是LINUX系统，那么......

别忘了使用“ -n ”选项。

```
$ pg var_test
#!/bin/sh
# var_test
echo "First Name :\c"
```

China-pub.com 下载

(续)

```
read name
echo "Middle Name :\c"
read middle
echo "Last name :\c"
read surname
```

5.3 cat

cat是一个简单而通用的命令，可以用它来显示文件内容，创建文件，还可以用它来显示控制字符。在使用cat命令时要注意，它不会在文件分页符处停下来；它会一下显示完整个文件。如果希望每次显示一页，可以使用 more命令或把cat命令的输出通过管道传递到另外一个具有分页功能的命令中，请看下面的例子：

```
$ cat myfile | more
```

或

```
$ cat myfile | pg
```

cat命令的一般形式为：

```
cat [options] filename1 ... filename2 ...
```

cat命令最有用的选项就是：

-v 显示控制字符

如果希望显示名为myfile的文件，可以用：

```
$ cat myfile
```

如果希望显示myfile1、myfile2、myfile3这三个文件，可以用：

```
$ cat myfile1 myfile2 myfile3
```

如果希望创建一个名为bigfile的文件，该文件包含上述三个文件的内容，可以把上面命令的输出重定向到新文件中：

```
$ cat myfile1 myfile2 myfile3 > bigfile
```

如果希望创建一个新文件，并向其中输入一些内容，只需使用 cat命令把标准输出重定向到该文件中，这时 cat命令的输入是标准输入 —— 键盘，你输入一些文字，输入完毕后按<CTRL-D>结束输入。这真是一个非常简单的文字编辑器！

```
$ cat > myfile
This is great
<CTRL-D>
$ pg myfile
This is great
```

还可以使用cat命令来显示控制字符。这里有一个对从 DOS机器上ftp过来的文件进行检察的例子，在这个例子中，所有的控制字符 <CTRL-M>都在行末显示了出来。

```
$ cat -v life.tct
ERROR ON REC AS12^M
ERROR ON REC AS31^M
```

有一点要提醒的是，如果在敲入了 cat以后就直接按回车，该命令会等你输入字符。如果你本来就是要输入一些字符，那么它除了会在你输入时在屏幕上显示以外，还会再回显这些

China-pub.com 下载

内容；最后按<CTRL-D>结束输入即可。

5.4 管道

可以通过管道把一个命令的输出传递给另一个命令作为输入。管道用竖杠 |表示。它的一般形式为：

命令1 |命令2

其中|是管道符号。

在下面的例子中，在当前目录中执行文件列表操作，如果没有管道的话，所有文件就会显示出来。当shell看到管道符号以后，就会把所有列出的文件交给管道右边的命令，因此管道的含义正如它的名字所暗示的那样：把信息从一端传送到另外一端。在这个例子中，接下来grep命令在文件列表中搜索quarter1.doc：

```
$ ls | grep quarter1.doc
quarter1.doc
```

让我们再来用一幅图形象地讲解刚才的例子（见图 5-1）：

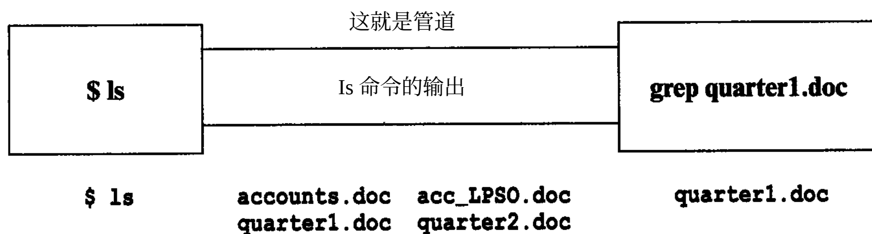


图5-1 管道

sed、awk和grep都很适合用管道，特别是在简单的一行命令中。在下面的例子中， who命令的输出通过管道传递给awk命令，以便只显示用户名和所在的终端。

```
$ who | awk '{print $1"\t"$2}'
matthew     pts/0
louise      pts/1
```

如果你希望列出系统中所有的文件系统，可以使用管道把 df命令的输出传递给awk命令，awk显示出其中的第一列。你还可以再次使用管道把 awk的结果传递给grep命令，去掉最上面的题头filesystem。

```
$ df -k | awk '{print $1}'| grep -v "Filesystem"
/dev/hda5
/dev/hda8
/dev/hda6
/dev/hdb5
/dev/hdb1
/dev/hda7
/dev/hda1
```

当然，你没准还会希望只显示出其中的分区名，不显示 /dev/部分，这没问题；我们只要在后面简单地加上另一个管道符号和相应的 sed命令即可。

```
$ df -k | awk '{print $1}'| grep -v "Filesystem"|sed s'/\/dev\///g'
hda5
```

China-pub.com
下载


```
hda8
hda6
hdb5
hdb1
hda7
hda1
```

在这个例子中，我们先对一个文件进行排序，然后通过管道输送到打印机。

```
$ sort myfile | lp
```

5.5 tee

tee命令作用可以用字母T来形象地表示。它把输出的一个副本输送到标准输出，另一个副本拷贝到相应的文件中。如果希望在看到输出的同时，也将其存入一个文件，那么这个命令再合适不过了。

它的一般形式为：

```
tee -a files
```

其中，-a表示追加到文件末尾。

当执行某些命令或脚本时，如果希望把输出保存下来，tee命令非常方便。

下面我们来看一个例子，我们使用who命令，结果输出到屏幕上，同时保存在who.out文件中：

```
$ who | tee who.out
louise    pts/1    May 20 12:58 (193.132.90.9)
matthew   pts/0    May 20 10:18 (193.132.90.1)

cat who.out
louise    pts/1    May 20 12:58 (193.132.90.9)
matthew   pts/0    May 20 10:18 (193.132.90.1)
```

可以用图5-2来表示刚才的例子。



图5-2 tee

在下面的例子中，我们把一些文件备份到磁带上，同时将所备份的文件记录在tape.log文件中。由于需要不断地对文件进行备份，为了保留上一次的日志，我们在tee命令中使用了-a选项。

```
$ find  etc usr/local home -depth -print | cpio -ovC65536 -O \
/dev/rmt/0n | tee -a tape.log
```

在上面的例子中，第一行末尾的反斜杠\告诉shell该命令尚未结束，应从下面一行继续读入该命令。

可以在执行脚本之前，使用一个echo命令告诉用户谁在执行这个脚本，输出结果保存在

China-pub.com 下载

什么地方。

```
$ echo " myscript is now running, check out any errors...in
myscript.log" | tee -a myscript.log

$ myscript | tee -a myscript.log
```

如果不想把输出重定向到文件中，可以不这样做，而是把它定向到某个终端上。在下面的例子中，一个警告被发送到系统控制台上，表明一个磁盘清理进程即将运行。

```
$ echo "stand-by disk cleanup starting in 1 minute"| tee /dev/console
```

可以让不同的命令使用同一个日志文件，不过不要忘记使用 -a选项。

```
$ sort myfile | tee -a accounts.log

$ myscript | tee -a accounts.log
```

5.6 标准输入、输出和错误

当我们在shell中执行命令的时候，每个进程都和三个打开的文件相联系，并使用文件描述符来引用这些文件。由于文件描述符不容易记忆，shell同时也给出了相应的文件名。

下面就是这些文件描述符及它们通常所对应的文件名：

文 件	文件描述符
输入文件 —— 标准输入	0
输出文件 —— 标准输出	1
错误输出文件 —— 标准错误	2

系统中实际上有12个文件描述符，但是正如我们在上表中所看到的，0、1、2是标准输入、输出和错误。可以任意使用文件描述符 3到9。

5.6.1 标准输入

标准输入是文件描述符0。它是命令的输入，缺省是键盘，也可以是文件或其他命令的输出。

5.6.2 标准输出

标准输出是文件描述符1。它是命令的输出，缺省是屏幕，也可以是文件。

5.6.3 标准错误

标准错误是文件描述符2。这是命令错误的输出，缺省是屏幕，同样也可以是文件。你可能会问，为什么会有一个专门针对错误的特殊文件？这是由于很多人喜欢把错误单独保存到一个文件中，特别是在处理大的数据文件时，可能会产生很多错误。

如果没有特别指定文件说明符，命令将使用缺省的文件说明符（你的屏幕，更确切地说是你的终端）。

5.7 文件重定向

在执行命令时，可以指定命令的标准输入、输出和错误，要实现这一点就需要使用文件

China-pub.com 下载

重定向。表5-1列出了最常用的重定向组合，并给出了相应的文件描述符。

在对标准错误进行重定向时，必须要使用文件描述符，但是对于标准输入和输出来说，这不是必需的。为了完整起见，我们在表5-1中列出了两种方法。

表5-1 常用文件重定向命令

command > filename	把标准输出重定向到一个新文件中
command >> filename	把标准输出重定向到一个文件中(追加)
command 1 > fielname	把标准输出重定向到一个文件中
command > filename 2>&1	把标准输出和标准错误一起重定向到一个文件中
command 2 > filename	把标准错误重定向到一个文件中
command 2 >> filename	把标准输出重定向到一个文件中(追加)
command >> filename 2>&1	把标准输出和标准错误一起重定向到一个文件中(追加)
command < filename >filename2	command命令以filename文件作为标准输入，以filename2文件作为标准输出
command < filename	command命令以filename文件作为标准输入
command << delimiter	从标准输入中读入，直至遇到delimiter分界符
command <&m	把文件描述符m作为标准输入
command >&m	把标准输出重定向到文件描述符m中
command <&-	关闭标准输入

5.7.1 重定向标准输出

让我们来看一个标准输出的例子。在下面的命令中，把 /etc/passwd文件中的用户ID域按照用户命排列。该命令的输出重定向到sort.out文件中。要提醒注意的是，在使用sort命令的时候(或其他含有相似输入文件参数的命令)，重定向符号一定要离开sort命令两个空格，否则该命令会把它当作输入文件。

```
$ cat passwd | awk -F: '{print $1}' | sort  1>sort.out
```

从表5-1中可以看出，我们也可以使用如下的表达方式，结果和上面一样：

```
$ cat passwd | awk -F: '{print $1}' | sort  >sort.out
```

可以把很多命令的输出追加到同一文件中。

```
$ ls -l | grep ^d >>files.out
$ ls account* >> files.out
```

在上面的例子中，所有的目录名和以account开头的文件名都被写入到file.out文件中。

如果希望把标准输出重定向到文件中，可以用 >filename。在下面的例子中，ls命令的所有输出都被重定向到ls.out文件中：

```
$ ls >ls.out
```

如果希望追加到已有的文件中（在该文件不存在的情况下创建该文件），那么可以使用>>filename：

```
$ pwd >>path.out
$ find . -name "LPSO.doc" -print >>path.out
```

如果想创建一个长度为0的空文件，可以用'>filename'：

```
$ >myfile
```

China-pub.com 下载

5.7.2 重定向标准输入

可以指定命令的标准输入。在awk一章就会遇到这样的情况。下面给出一个这样的例子：

```
$ sort  < name.txt
```

在上面的命令中，sort命令的输入是采用重定向的方式给出的，不过也可以直接把相应的文件作为该命令的参数：

```
$ sort name.txt
```

在上面的例子中，还可以更进一步地通过重定向为sort命令指定一个输出文件name.out。这样屏幕上将不会出现任何信息(除了错误信息以外)：

```
$ sort  <name.txt >name.out
```

在发送邮件时，可以用重定向的方法发送一个文件中的内容。在下面的例子中，用户louise将收到一个邮件，其中含有文件contents.txt中的内容：

```
$ mail louise < contents.txt
```

重定向操作符command << delimiter是一种非常有用的命令，通常都被称为“此处”文挡。我们将在本书后面的章节深入讨论这一问题。现在只介绍它的功能。shell将分界符delimiter之后直至下一个同样的分界符之前的所有内容都作为输入，遇到下一个分界符，shell就知道输入结束了。这一命令对于自动或远程的例程非常有用。可以任意定义分界符delimiter，最常见的是EOF，而我最喜欢用MAYDAY，这完全取决于个人的喜好。还可以在<<后面输入变量。下面给出一个例子，我们创建了一个名为myfile的文件，并在其中使用了TERM和LOGNAME变量。

```
$ cat >> myfile <<MAYDAY
> Hello there I am using a $TERM terminal
> and my user name is $LOGNAME
> bye...
> MAYDAY

$ pg myfile
Hello there I am using a vt100 terminal
and my user name is dave
bye...
```

5.7.3 重定向标准错误

为了重定向标准错误，可以指定文件描述符2。让我们先来看一个例子，因为举例子往往会让人更容易明白。在这个例子中，grep命令在文件missiles中搜索trident字符串：

```
$ grep "trident" missiles
grep: missiles: No such file or directory
```

grep命令没有找到该文件，缺省地向终端输出了一个错误信息。现在让我们把错误重定向到文件/dev/null中(实际就上是系统的垃圾箱)：

```
$ grep "trident" missiles 2>/dev/null
```

这样所有的错误输出都输送到了/dev/null，不再出现在屏幕上。

如果你在对更重要的文件进行操作，可能会希望保存相应的错误。下面就是一个这样的例子，这一次错误被保存到grep.err文件中：

```
$  grep "trident" missiles 2>grep.err
$ pg grep.err
grep: missiles: No such file or directory
```

还可以把错误追加到一个文件中。在使用一组命令完成同一个任务时，这种方法非常有用。在下面的例子中，两个 grep命令把错误都输出到同一个文件中；由于我们使用了 >>符号进行追加，后面一个命令的错误(如果有的话)不会覆盖前一个命令的错误。

```
$ grep "LPSO" * 2>>account.err
$ grep "SILO" * 2>>account.err
```

5.8 结合使用标准输出和标准错误

一个快速发现错误的方法就是，先将输出重定向到一个文件中，然后再把标准错误重定向到另外一个文件中。下面给出一个例子：

我有两个审计文件，其中一个的确存在，而且包含一些信息，而另一个由于某种原因已经不存在了(但我不知道)。我想把这两个文件合并到accounts.out文件中。

```
$ cat account_qtr.doc account_end.doc 1>accounts.out 2>accounts.err
```

现在如果出现了错误，相应的错误将会保存在 accounts.err文件中。

```
$ pg accounts.out
AVBD 34HJ   OUT
AVFJ   31KO  OUT
...

$ pg accounts.err
cat: account_end.doc: No such file or directory
```

我事先并不知道是否存在 account_end.doc文件，使用上面的方法能够快速发现其中的错误。

5.9 合并标准输出和标准错误

在合并标准输出和标准错误的时候，切记 shell是从左至右分析相应的命令的。下面给出一个例子：

```
$ cleanup >cleanup.out 2>&1
```

在上面的例子中，我们将 cleanup脚本的输出重定向到 cleanup.out文件中，而且其错误也被重定向到相同的文件中。

```
$ grep "standard"* > grep.out 2>&1
```

在上面的例子中， grep命令的标准输出和标准错误都被重定向到 grep.out文件中。你在使用前面提到的“此处”文挡时，有可能需要把所有的输出都保存到一个文件中，这样万一出现了错误，就能够被记录下来。通过使用 2>&1就可以做到这一点，下面给出一个例子：

```
$ cat>> filetest 2>&1 <<MAYDAY
> This is my home $HOME directory
> MAYDAY
$ pg filetest
This is my home /home/dave directory
```

上面的例子演示了如何把所有的输出捕捉到一个文件中。在使用 cat命令的时候，这可能

China-pub.com 下载

没什么用处，不过如果你使用“此处”文挡连接一个数据库管理系统 (例如使用isql连接sybase)或使用ftp，这一点就变得非常重要了，因为这样就可以捕捉到所有的错误，以免这些错误在屏幕上一闪而过，特别是在你不在的时候。

5.10 exec

exec命令可以用来替代当前shell；换句话说，并没有启动子shell。使用这一命令时任何现有环境都将会被清除，并重新启动一个shell。它的一般形式为：

```
exec command
```

其中的command通常是一个shell脚本。

我所能够想像得出的描述exec命令最贴切的说法就是：它践踏了你当前的shell。

当这个脚本结束时，相应的会话可能就结束了。 exec命令的一个常见用法就是在用户的.profile最后执行时，用它来执行一些用于增强安全性的脚本。如果用户的输入无效，该shell将被关闭，然后重新回到登录提示符。 exec还常常被用来通过文件描述符打开文件。

记住， exec在对文件描述符进行操作的时候（也只有在这时），它不会覆盖你当前的shell。

5.11 使用文件描述符

可以使用exec命令通过文件描述符打开和关闭文件。在下面的例子中，我选用了文件描述符4，实际上我可以在4到9之间任意选择一个数字。下面的脚本只是从stock.txt文件中读了两行，然后把这两行回显出来。

该脚本的第一行把文件描述符4指定为标准输入，然后打开stock.txt文件。接下来两行的作用是读入了两行文本。接着，作为标准输入的文件描述符4被关闭。最后，line1和line2两个变量所含有的内容被回显到屏幕上。

```
$ pg f_desc
#!/bin/sh
# f_desc
exec 4<&0 0<stock.txt
read line1
read line2
exec  0<&4
echo $line1
echo $line2
```

下面是这个小小的股票文件stock.txt的内容：

```
$ pg stock.txt
Crayons Assorted 34
Pencils Light    12
```

下面是该脚本的运行结果：

```
$ f_desc
Crayons Assorted 34
Pencils Light    12
```

上面是一个关于文件描述符应用的简单例子。它看起来没有什么用处。在以后讲解循环的时候，将会给出一个用文件描述符代替cp命令拷贝文本文件的例子。

China-pub.com
下载

5.12 小结

本书通篇可见重定向的应用，因为它是 shell中的一个重要部分。通过重定向，可以指定命令的输入；如果有错误的话，可以用一个单独的文件把它们记录下来，这样就可以方便快捷地查找问题。

这里没有涉及的就是文件描述符的应用 (3～9)。要想应用这些文件描述符，就一定会涉及循环方法，在后面讲到循环方法的时候，我们会再次回过头来讲述有关文件描述符的问题。

China-pub.com
下载

第6章 命令执行顺序

在执行某个命令的时候，有时需要依赖于前一个命令是否执行成功。例如，假设你希望将一个目录中的文件全部拷贝到另外一个目录中后，然后删除源目录中的全部文件。在删除之前，你希望能够确信拷贝成功，否则就有可能丢失所有的文件。

在本章中，我们将讨论：

• 命令执行控制。

• 命令组合。

如果希望在成功地执行一个命令之后再执行另一个命令，或者在一个命令失败后再执行另一个命令，&&和||可以完成这样的功能。相应的命令可以是系统命令或shell脚本。

Shell还提供了在当前shell或子shell中执行一组命令的方法，即使用（）和{}。

6.1 使用&&

使用&&的一般形式为：

```
命令1 && 命令2
```

这种命令执行方式相当地直接。&&左边的命令（命令1）返回真(即返回0，成功被执行）后，&&右边的命令（命令2）才能够被执行；换句话说，“如果这个命令执行成功&&那么执行这个命令”。

这里有一个使用&&的简单例子：

```
$ cp justice.doc justice.bak && echo "if you are seeing this then cp was
OK"

if you are seeing this then cp was OK
```

在上面的例子中，&&前面的拷贝命令执行成功，所以&&后面的命令（echo命令）被执行。

再看一个更为实用的例子：

```
$ mv /apps/bin /apps/dev/bin && rm -r /apps/bin
```

在上面的例子中，/apps/bin目录将会被移到/apps/dev/bin目录下，如果它没有被成功执行，就不会删除/apps/bin目录。

在下面的例子中，文件quarter_end.txt首先将被排序并输出到文件quarter.sorted中，只有这一命令执行成功之后，文件quarter.sorted才会被打印出来：

```
$ sort quarter_end.txt > quarter.sorted && lp quarter.sorted
```

6.2 使用||

使用||的一般形式为：

```
命令1 || 命令2
```

||的作用有一些不同。如果||左边的命令（命令1）未执行成功，那么就执行||右边的命令（命令2）；或者换句话说，“如果这个命令执行失败了||那么就执行这个命令”。

这里有一个使用||的简单例子：

```
$ cp wopper.txt oops.txt || echo "if you are seeing this cp failed"

cp: wopper.txt: No such file or directory
if you are seeing this cp failed
```

在上面的例子中，拷贝命令没有能够被成功执行，因此||后面的命令被执行。

这里有一个更为实用的例子。我希望从一个审计文件中抽取第1个和第5个域，并将其输出到一个临时文件中，如果这一操作未成功，我希望能够收到一个相应邮件：

```
$ awk '{print$1,$5}' acc.qtr >qtr.tmp || echo "Sorry the payroll
extraction didn't work" | mail dave
```

在这里不只可以使用系统命令；这里我们首先对month_end.txt文件执行了一个名为comet的shell脚本，如果该脚本未执行成功，该shell将结束。

```
$ comet month_end.txt || exit
```

6.3 用（）和{ }将命令结合在一起

如果希望把几个命令合在一起执行，shell提供了两种方法。既可以在当前shell也可以在子shell中执行一组命令。

为了在当前shell中执行一组命令，可以用命令分隔符隔开每一个命令，并把所有的命令用圆括号（）括起来。

它的一般形式为：

```
(命令1;命令2;...)
```

如果使用{}来代替（），那么相应的命令将在子shell而不是当前shell中作为一个整体被执行，只有在{}中所有命令的输出作为一个整体被重定向时，其中的命令才被放到子shell中执行，否则在当前shell执行。它的一般形式为：

```
{命令1;命令2;...}
```

我很少单独使用这两种方法。我一般只和&&或||一起使用这两种方法。

再回到前面那个comet脚本的例子，如果这个脚本执行失败了，我很可能会希望执行两个以上的命令，而不只是一个命令。我可以使用这两种方法。这是原先那个例子：

```
$ comet month_end.txt || exit
```

现在如果该脚本执行失败了，我希望先给自己发个邮件，然后再退出，可以用下面的方法来实现：

```
$ comet month_end || (echo "Hello, guess what! Comet did not work"|mail
dave; exit)
```

在上面的例子中，如果只使用了命令分隔符而没有把它们组合在一起，shell将直接执行最后一个命令（exit）。

我们再回头来看看前面那个使用&&排序的例子，下面是原来的那个例子：

```
$ sort quarter_end.txt > quarter.sorted && lp quarter.sorted
```

使用命令组合的方法，如果sort命令执行成功了，可以先将输出文件拷贝到一个日志区，

China-pub.com
下载

然后再打印。

```
$ sort quarter_end.txt > quarter.sorted && ( cp quarter.sorted /logs/
quarter.sorted; lp quarter.sorted )
```

6.4 小结

在编写shell脚本时，使用&&和||对构造判断语句非常有用。如果希望在前一个命令执行失败的情况不执行后面的命令，那么本章所讲述的方法非常简单有效。使用这样的方法，可以根据&&或||前面命令的返回值来控制其后面命令的执行。

China-pub.com
下载

第二部分 文 本 过 滤

第7章 正则表达式介绍

随着对UNIX和LINUX熟悉程度的不断加深，需要经常接触到正则表达式这个领域。使用shell时，从一个文件中抽取多于一个字符串将会很麻烦。例如，在一个文本中抽取一个词，它的头两个字符是大写的，后面紧跟四个数字。如果不使用某种正则表达式，在 shell中将不能实现这个操作。

本章内容包括：

- 匹配行首与行尾。
- 匹配数据集。
- 只匹配字母和数字。
- 匹配一定范围内的字符串集。

当从一个文件或命令输出中抽取或过滤文本时，可以使用正则表达式（ RE），正则表达式是一些特殊或不很特殊的字符串模式的集合。

为了抽取或获得信息，我们给出抽取操作应遵守的一些规则。这些规则由一些特殊字符或进行模式匹配操作时使用的元字符组成。也可以使用规则字符作为模式中的一部分进行搜寻。例如， A将查询A，x将查找字母x。

系统自带的所有大的文本过滤工具在某种模式下都支持正则表达式的使用，并且还包括一些扩展的元字符集。这里只涉及其中之一，即以字符出现情况进行匹配的表达式，原因是一些系统将这类模式划分为一组形成基本元字符的集合。这是一个好想法，本书也采用这种方式。

本章设计的基本元字符使用在grep和sed命令中，同时结合{\\}（以字符出现情况进行匹配的元字符）使用在awk语言中。

表7-1 基本元字符集及其含义

^	只匹配行首
$	只匹配行尾
*	一个单字符后紧跟*，匹配0个或多个此单字符
[ ]	匹配[]内字符。可以是一个单字符，也可以是字符序列。可以使用 -表示[]内字符序列范围，如用[1-5]代替[12345]
\	用来屏蔽一个元字符的特殊含义。因为有时在 shell中一些元字符有特殊含义。\可以使其失去应有意义
.	匹配任意单字符
pattern\{n\}	用来匹配前面pattern出现次数。n为次数
pattern\{n，\}m	含义同上，但次数最少为n
pattern\{n，m\}	含义同上，但pattern出现次数在n与m之间

现在详细讲解其中特殊含义。

China-pub.com 下载

7.1 使用句点匹配单字符

句点“.”可以匹配任意单字符。例如，如果要匹配一个字符串，以beg开头，中间夹一个任意字符，那么可以表示为beg.n，“.”可以匹配字符串头，也可以是中间任意字符。

在ls -l命令中，可以匹配一定权限：

```
...x..x..x
```

此格式匹配用户本身，用户组及其他组成员的执行权限。

```
drwxrwxrw-    – no match
-rw-rw-rw-    – no match
-rwx-rwxr-x   – match
-rwx-r-x-r-x  – match
```

假定正在过滤一个文本文件，对于一个有10个字符的脚本集，要求前4个字符之后为XC，匹配操作如下：

```
....XC....
```

以上例子解释为前4个字符任意，5，6字符为XC，后4个字符也任意，按下例运行：

```
1234XC9088   – match
4523XX9001   – no match
0011XA9912   – no match
9931XC3445   – match
```

注意，“.”允许匹配ASCII集中任意字符，或为字母，或为数字。

7.2 在行首以^匹配字符串或字符序列

^只允许在一行的开始匹配字符或单词。例如，使用ls -l命令，并匹配目录。之所以可以这样做是因为ls -l命令结果每行第一个字符是d，即代表一个目录。

```
^d
drwxrwxrw-    – match
-rw-rw-rw-    – no match
drwx-rwxr-x   – match
-rwx-r-x-r-x  – no match
```

回到脚本（1），使用^001，结果将匹配每行开始为001的字符串或单词：

```
1234XC9088   – no match
4523XX9001   – no match
0011XA9912   – match
9931XC3445   – no match
```

可以将各种模式结合使用，例如：

```
^...4XC....
```

结果为：

```
1234XC9088   – match
4523XX9001   – no match
0011XA9912   – no match
9931XC3445   – no match
3224XC193    – no match
```

China-pub.com 下载

以上模式表示，在每行开始，匹配任意 3个字符，后跟4XC，最后为任意4个字符。^在正则表达式中使用频繁，因为大量的抽取操作通常在行首。

在行首第4个字符为1，匹配操作表示为：

```
^...1
```

结果为：

```
1234XC9088    – no match
4523XX9001    – no match
0011XA9912    – match
9931XC3445    – match
```

行首前4个字符为comp，匹配操作表示为：

```
^comp
```

假定重新定义匹配模式，行首前 4个字符为comp，后面紧跟两个任意字符，并以 ing结尾，一种方法为：

```
^comp..ing
```

以上例子太明显了，不是很有用，但仍讲述了混合使用正则模式的基本概念。

7.3 在行尾以$匹配字符串或字符

可以说$与^正相反，它在行尾匹配字符串或字符， $符号放在匹配单词后。假定要匹配以单词trouble结尾的所有行，操作为：

```
trouble$
```

类似的，使用1d$返回每行以1d结尾的所有字符串。

如果要匹配所有空行，执行以下操作：

```
^$
```

具体分析为匹配行首，又匹配行尾，中间没有任何模式，因此为空行。

如果只返回包含一个字符的行，操作如下：

```
^.$
```

不像空白行，在行首与行尾之间有一个模式，代表任意单字符。

如果在行尾匹配单词jet01，操作如下：

```
jet01$
```

7.4 使用*匹配字符串中的单字符或其重复序列

使用此特殊字符匹配任意字符或字符串的重复多次表达式。例如：

```
compu*t
```

将匹配字符u一次或多次：

```
computer
computing
compuuuute
```

另一个例子：

```
10133*
```

匹配

China-pub.com 下载

```
101333
10133
101344444
```

7.5 使用\屏蔽一个特殊字符的含义

有时需要查找一些字符或字符串，而它们包含了系统指定为特殊字符的一个字符。什么是特殊字符？一般意义上讲，下列字符可以认为是特殊字符：

```
$ . ' " * [ ] ^ | () \ + ?
```

假定要匹配包含字符“ . ”的各行而“ , ”代表匹配任意单字符的特殊字符，因此需要屏蔽其含义。操作如下：

```
\.
```

上述模式不认为反斜杠后面的字符是特殊字符，而是一个普通字符，即句点。

假定要匹配包含^的各行，将反斜杠放在它前面就可以屏蔽其特殊含义。如下：

```
\^
```

如果要在正则表达式中匹配以*.pas结尾的所有文件，可做如下操作：

```
\*\.pas
```

即可屏蔽字符*的特定含义。

7.6 使用[]匹配一个范围或集合

使用[]匹配特定字符串或字符串集，可以用逗号将括弧内要匹配的不同字符串分开，但并不强制要求这样做（一些系统提倡在复杂的表达式中使用逗号），这样做可以增加模式的可读性。

使用“ - ”表示一个字符串范围，表明字符串范围从“ - ”左边字符开始，到“ - ”右边字符结束。

如果熟知一个字符串匹配操作，应经常使用 []模式。

假定要匹配任意一个数字，可以使用：

```
[0123456789]
```

然而，通过使用“ - ”符号可以简化操作：

```
[0-9]
```

或任意小写字母

```
[a-z]
```

要匹配任意字母，则使用：

```
[A-Za-z]
```

表明从A-Z、a-z的字母范围。

如要匹配任意字母或数字，模式如下：

```
[A-Za-z0-9]
```

在字符序列结合使用中，可以用 []指出字符范围。假定要匹配一单词，以 s开头，中间有一任意字母，以t结尾，那么操作如下：

```
s[a-z A-Z]t
```

上述过程返回大写或小写字母混合的单词，如仅匹配小写字母，可使用：

```
s[a-z]t
```

如要匹配Computer或computer两个单词，可做如下操作：

```
[Cc]omputer
```

为抽取诸如Scout、shout、bought等单词，使用下列表达式：

```
[ou] .*t
```

匹配以字母o或u开头，后跟任意一个字符任意次，并以t结尾的任意字母。

也许要匹配所有包含system后跟句点的所有单词，这里S可大写或小写。使用如下操作：

```
[S,s]ystem\.
```

[]在指定模式匹配的范围或限制方面很有用。结合使用 *与[]更是有益，例如[A-Za-Z]*将匹配所有单词。

```
[A-Za-z]*
```

注意^符号的使用，当直接用在第一个括号里，意指否定或不匹配括号里内容。

```
[^a-zA-Z]
```

匹配任一非字母型字符，而

```
[^0-9]
```

匹配任一非数字型字符。

通过最后一个例子，应可猜知除了使用 ^，还有一些方法用来搜索任意一个特殊字符。

7.7 使用\{\}匹配模式结果出现的次数

使用*可匹配所有匹配结果任意次，但如果只要指定次数，就应使用 \{\}，此模式有三种形式，即：

pattern\{n\}　匹配模式出现n次。

pattern\{n,\}　匹配模式出现最少n次。

pattern\{n,m}　匹配模式出现n到m次之间，n,m为0-255中任意整数。

请看第一个例子，匹配字母A出现两次，并以B结尾，操作如下：

```
A\{2\}B
```

匹配值为AAB

匹配A至少4次，使用：

```
A\{4,\}B
```

可以得结果AAAAB或AAAAAAAB，但不能为AAAB。

如给出出现次数范围，例如A出现2次到4次之间：

```
A\{2,4\}B
```

则结果为AAB、AAAB、AAAAB，而不是AB或AAAAAB等。

假定从下述列表中抽取代码：

1234XC9088
4523XX9001
0011XA9912
9931XC3445

格式如下：前4个字符是数字，接下来是xx，最后4个也是数字，操作如下：

China-pub.com 下载

```
[0-9]\{4\}XX[0-9]\{4\}
```

具体含义如下：

1) 匹配数字出现4次。

2) 后跟代码xx。

3) 最后是数字出现4次。

结果为：

```
1234XC9088   – no match
4523XX9001   – match
0011XA9912   – no match
9931XC3445   – no match
```

在写正则表达式时，可能会有点难度或达不到预期效果，一个好习惯是在写真正的正则表达式前先写下预期的输出结果。这样做，当写错时，可以逐渐修改，以消除意外结果，直至返回正确值。为节省设计基本模式的时间，表 7-2给出一些例子，这些例子并无特别顺序。

表7-2 经常使用的正则表达式举例

^	行首
$	行尾
^[the]	以the开头行
[Ss]igna[lL]	匹配单词signal、signaL、Signal、SignaL
[Ss]igna[lL]\.	同上，但加一句点
[mayMAY]	包含may大写或小写字母的行
^USER$	只包含USER的行
[tty]$	以tty结尾的行
\.	带句点的行
^d..x..x..x	对用户、用户组及其他用户组成员有可执行权限的目录
^[^l]	排除关联目录的目录列表
[.*0]	0之前或之后加任意字符
[000*]	000或更多个
[iI]	大写或小写I
[iI][nN]	大写或小写i或n
[^$]	空行
[^.*$]	匹配行中任意字符串
^......$	包括6个字符的行
[a- zA-Z]	任意单字符
[a-z][a-z]*	至少一个小写字母
[^0-9\$]	非数字或美元标识
[^0-0A-Za-z]	非数字或字母
[123]	1到3中一个数字
[Dd]evice	单词device或Device
De..ce	前两个字母为De，后跟两个任意字符，最后为ce

China-pub.com
下载

（续）

\^q	以^q开始行
^.$	仅有一个字符的行
^\.[0-9][0-9]	以一个句点和两个数字开始的行
'"Device"'	单词device
De[Vv]ice\.	单词Device或device
[0-9]\{2\}-[0-9]\{2\}-[0-9]\{4\}	日期格式dd-mm-yyyy
[0-9]\{3\}\.[0-9]\{3\}\.[0-9]\{3\}\.[0-9]\{3\}	IP地址格式nnn. nnn.nnn.nnn
[^.*$]	匹配任意行

7.8 小结

在shell编程中，一段好的脚本与完美的脚本间的差别之一，就是要熟知正则表达式并学会使用它们。相比较起来，用一个命令抽取一段文本比用三四个命令得出同样的结果要节省许多时间。

既然已经学会了正则表达式中经常使用的基本特殊字符，又通过一些例子简化了其复杂操作，那么现在可以看一些真正的例程了。

好，下面将讲述大量的grep,sed和awk例程。

China-pub.com
下载

第8章 grep 家 族

相信grep是UNIX和LINUX中使用最广泛的命令之一。 grep（全局正则表达式版本）允许对文本文件进行模式查找。如果找到匹配模式， grep打印包含模式的所有行。 grep支持基本正则表达式，也支持其扩展集。 grep有三种变形，即：

Grep：标准grep命令，本章大部分篇幅集中讨论此格式。

Egrep：扩展grep，支持基本及扩展的正则表达式，但不支持 \q模式范围的应用，与之相对应的一些更加规范的模式，这里也不予讨论。

Fgrep：快速grep。允许查找字符串而不是一个模式。不要误解单词 fast，实际上它与grep速度相当。

在本章中我们将讨论：

- grep（参数）选项。
- 匹配grep的一般模式。
- 只匹配字母或数字，或两者混用。
- 匹配字符串范围。

实际上应该只有一个grep命令，但不幸的是没有一种简单形式能够统一处理 grep的三种变形，将之合而为一，并保持 grep单模式处理时的速度。 GNU grep虽然在融合三种变形上迈进了一大步，但仍不能区分元字符的基本集和扩展集。上一章只讨论了基本的正则表达式，但在查看grep时也涉及到一些扩展模式的匹配操作。然而，首先还是先讨论一下在 grep和fgrep及egrep中均可使用的grep模式吧。

开始讨论之前，先生成一个文件，插入一段文本，并在每列后加入 <Tab>键，grep命令示例中绝大多数将以此为例，其命名为 data..f。生成一个文件，但不知其含义，将是一件很枯燥的事。那么先来看看data.f的记录结构。

第1列：城市位置编号。

第2列：月份。

第3列：存储代码及出库年份。

第4列：产品代号。

第5列：产品统一标价。

第6列：标识号。

第7列：合格数量。

```
$ pg data.f
48    Dec    3BC1997    LPSX   68.00  LVX2A  138
483   Sept   5AP1996    USP    65.00  LVX2C  189
47    Oct    3ZL1998    LPSX   43.00  KVM9D  512
219   dec    2CC1999    CAD    23.00  PLV2C  68
484   nov    7PL1996    CAD    49.00  PLV2C  234
483   may    5PA1998    USP    37.00  KVM9D  644
216   sept   3ZL1998    USP    86.00  KVM9E  234
```

China-pub.com
下载

8.1 grep

grep一般格式为：

grep [选项]基本正则表达式[文件]

这里基本正则表达式可为字符串。

8.1.1 双引号引用

在grep命令中输入字符串参数时，最好将其用双引号括起来。例如："mystring"。这样做有两个原因，一是以防被误解为shell命令，二是可以用来查找多个单词组成的字符串，例如："jet plane"，如果不用双引号将其括起来，那么单词 plane将被误认为是一个文件，查询结果将返回"文件不存在"的错误信息。

在调用变量时，也应该使用双引号，诸如： grep"$MYVAR"文件名，如果不这样，将没有返回结果。

在调用模式匹配时，应使用单引号。

8.1.2 grep选项

常用的grep选项有：

-c 只输出匹配行的计数。

-i 不区分大小写（只适用于单字符)。

-h 查询多文件时不显示文件名。

-l 查询多文件时只输出包含匹配字符的文件名。

-n 显示匹配行及行号。

-s 不显示不存在或无匹配文本的错误信息。

-v 显示不包含匹配文本的所有行。

8.1.3 查询多个文件

如果要在当前目录下所有.doc文件中查找字符串"sort"，方法如下：

```
$ grep "sort"*.doc
```

或在所有文件中查询单词"sort it"

```
$ grep "sort it" *
```

现在讲述在文本文件中grep选项的用法。

8.1.4 行匹配

```
$ grep -c "48"data.f
$4
```

grep返回数字4，意义是有4行包含字符串"48"。

现在显示包含"48"字符串的4行文本：

```
$ grep "48" data.f
48      Dec     3BC1997   LPSX    68.00   LVX2A   138
483     Sept    5AP1996   USP     65.00   LVX2C   189
484     nov     7PL1996   CAD     49.00   PLV2C   234
```

China-pub.com 下载

```
483     may     5PA1998  USP    37.00   KVM9D   644
```

8.1.5 行数

显示满足匹配模式的所有行行数：

```
$ grep -n "48" data.f
1:48    Dec     3BC1997  LPSX   68.00   LVX2A   138
2:483   Sept    5AP1996  USP    65.00   LVX2C   189
5:484   Nov     7PL1996  CAD    49.00   PLV2C   234
6:483   May     5PA1998  USP    37.00   KVM9D   644
```

行数在输出第一列，后跟包含48的每一匹配行。

8.1.6 显示非匹配行

显示所有不包含48的各行：

```
$ grep -v "48" data.f
47      Oct     3ZL1998  LPSX   43.00   KVM9D   512
219     Dec     2CC1999  CAD    23.00   PLV2C    68
216     Sept    3ZL1998  USP    86.00   KVM9E   234
```

8.1.7 精确匹配

可能大家已注意到，在上一例中，抽取字符串“ 48 ”，返回结果包含诸如484和483等包含“ 48 ”的其他字符串，实际上应精确抽取只包含 48的各行。注意在每个匹配模式中抽取字符串后有一个<Tab>键，所以应操作如下：

```
$ grep "48<tab>" data.f
48      Dec     3BC1997  LPSX   68.00   LVX2A   138
```

<Tab>表示点击tab键。

使用grep抽取精确匹配的一种更有效方式是在抽取字符串后加 \>。假定现在精确抽取 48，方法如下：

```
$ grep '48\>' data.f
48      Dec     3BC1997  LPSX   68.00   LVX2A   138
```

8.1.8 大小写敏感

缺省情况下， grep是大小写敏感的，如要查询大小写不敏感字符串，必须使用 -i开关。在data.f文件中有月份字符Sept，既有大写也有小写，要取得此字符串大小写不敏感查询，方法如下：

```
$ grep -i "sept" data.f
483     Sept    5AP1996  USP    65.00   LVX2C   189
216     sept    3ZL1998  USP    86.00   KVM9E   234
```

8.2 grep和正则表达式

使用正则表达式使模式匹配加入一些规则，因此可以在抽取信息中加入更多选择。使用正则表达式时最好用单引号括起来，这样可以防止 grep中使用的专有模式与一些 shell命令的特殊方式相混淆。

8.2.1 模式范围

假定要抽取代码为484和483的城市位置，上一章中讲到可以使用 []来指定字符串范围，这里用48开始，以3或4结尾，这样抽出484或483。

```
$ grep '48[34]' data.f
483     Sept    5AP1996  USP    65.00   LVX2C   189
484     nov     7PL1996  CAD    49.00   PLV2C   234
483     may     5PA1998  USP    37.00   KVM9D   644
```

8.2.2 不匹配行首

如果要抽出记录，使其行首不是48，可以在方括号中使用 ^记号，表明查询在行首开始。

```
$ grep '^[^48]' data.f
219     dec     2CC1999 CAD     23.00   PLV2C   68
216     sept    3ZL1998 USP     86.00   KVM9E   234
```

8.2.3 设置大小写

使用-i开关可以屏蔽月份 Sept的大小写敏感，也可以用另一种方式。这里使用 []模式抽取各行包含Sept和sept的所有信息。

```
$ grep '[Ss]ept' data.f
483     Sept    5AP1996 USP     65.00   LVX2C   189
216     sept    3ZL1998 USP     86.00   KVM9E   234
```

如果要抽取包含 Sept的所有月份，不管其大小写，并且此行包含字符串 483，可以使用管道命令，即符号“ | ”左边命令的输出作为“ | ”右边命令的输入。举例如下：

```
$ grep '[Ss]ept' data.f | grep 483
483     Sept    5AP1996  USP    65.00   LVX2C   189
```

不必将文件名放在第二个grep命令中，因为其输入信息来自于第一个 grep命令的输出。

8.2.4 匹配任意字符

如果抽取以L开头，以D结尾的所有代码，可使用下述方法，因为已知代码长度为 5个字符：

```
$ grep 'K...D' data.f
47      Oct     3ZL1998  LPSX   43.00   KVM9D   512
483     may     5PA1998  USP    37.00   KVM9D   644
```

将上述代码做轻微改变，头两个是大写字母，中间两个任意，并以 C结尾：

```
$ grep '[A-Z][A-Z]..C' data.f
483     Sept    5AP1996  USP    65.00   LVX2C   189
219     dec     2CC1999  CAD    23.00   PLV2C    68
484     nov     7PL1996  CAD    49.00   PLV2C   234
```

8.2.5 日期查询

一个常用的查询模式是日期查询。先查询所有以 5开始以1996或1998结尾的所有记录。使用模式5..199[6,8]。这意味着第一个字符为5，后跟两个点，接着是199，剩余两个数字是6或8。

China-pub.com 下载

```
$ grep '5..199[6,8]' data.f
483      Sept     5AP1996  USP      65.00    LVX2C     189
483      may      5PA1998  USP      37.00    KVM9D     644
```

查询包含1998的所有记录的另外一种方法是使用表达式 [0-9]\{3\}[8]，含义是任意数字重复3次，后跟数字8，虽然这个方法不像上一个方法那么精确，但也有一定作用。

```
$ grep '[0-9]\ {3\}[8]' data.f
47       Oct      3ZL1998  LPSX     43.00    KVM9D     512
483      may      5PA1998  USP      37.00    KVM9D     644
216      sept     3ZL1998  USP      86.00    KVM9E     234
```

8.2.6 范围组合

必须学会使用[]抽取信息。假定要取得城市代码，第一个字符为任意字符，第二个字符在0到5之间，第三个字符在0到6之间，使用下列模式即可实现。

```
$  grep '[0-9][0-5][0-6]' data.f
48       Dec      3BC1997  LPSX     68.00    LVX2A     138
483      Sept     5AP1996  USP      65.00    LVX2C     189
47       Oct      3ZL1998  LPSX     43.00    KVM9D     512
219      dec      2CC1999  CAD      23.00    PLV2C      68
484      nov      7PL1996  CAD      49.00    PLV2C     234
483      may      5PA1998  USP      37.00    KVM9D     644
216      sept     3ZL1998  USP      86.00    KVM9E     234
```

这里返回很多信息，有想要的，也有不想要的。参照模式，返回结果是正确的，因此这里还需要细化模式，可以以行首开始，使用 ^符号：

```
$  grep '^[0-9][0-5][0-6]' data.f
216      sept     3ZL1998  USP      86.00    KVM9E     234
```

这样可以返回一个预期的正确结果。

8.2.7 模式出现机率

抽取包含数字4至少重复出现两次的所有行，方法如下：

```
$ grep '4\{2,\}' data.f
483      may      5PA1998  USP      37.00    KVM9D     644
```

上述语法指明数字4至少重复出现两次。

同样，抽取记录使之包含数字999（三个9），方法如下：

```
$ grep '9\{3,\}' data.f
219      dec      2CC1999  CAD      23.00    PLV2C      68
```

如果要查询重复出现次数一定的所有行，语法如下，数字 9重复出现两次：

```
$ grep '9\{2\}' data.f
```

有时要查询重复出现次数在一定范围内，比如数字或字母重复出现 2到6次，下例匹配数字8重复出现2到6次，并以3结尾：

```
$ grep '6\{2,6}3'  myfile
```

```
83            – no match
888883        – match
8884          – no match
```

```
88883          – match
```

8.2.8 使用grep匹配"与"或者"或"模式

grep命令加-E参数，这一扩展允许使用扩展模式匹配。例如，要抽取城市代码为 219或216，方法如下：

```
$ grep -E '219|216' data.f
219     dec     2CC1999   CAD     23.00    PLV2C    68
216     sept    3ZL1998   USP     86.00    KVM9E   234
```

8.2.9 空行

结合使用^和$可查询空行。使用-n参数显示实际行数：

```
$ grep '^$' myfile
```

8.2.10 匹配特殊字符

查询有特殊含义的字符，诸如$. ' " * [] ^ | \ + ? ,必须在特定字符前加\。假设要查询包含"."的所有行，脚本如下：

```
$ grep '\.' myfile
```

或者是一个双引号：

```
$ grep '\"' myfile
```

以同样的方式，如要查询文件名conftroll.conf（这是一个配置文件），脚本如下：

```
$ grep 'conftroll\.conf' myfile
```

8.2.11 查询格式化文件名

使用正则表达式可匹配任意文件名。系统中对文本文件有其标准的命名格式。一般最多六个小写字符，后跟句点，接着是两个大写字符。例如，要在一个包含各类文件名的文件filename.deposit中定位这类文件名，方法如下：

```
$ grep '[^a-z]\{1,6/}\.[^A-Z]\{1,2/}' filename.deposit
yrend.AS       – match
mothdf         – nomatch
soa.PP         – match
qp.RR          – match
```

8.2.12 查询IP地址

查询DNS服务是日常工作之一，这意味着要维护覆盖不同网络的大量 IP地址。有时地址IP会超过2000个。如果要查看nnn.nnn网络地址，但是却忘了第二部分中的其余部分，只知有两个句点，例如nnn.nn..。要抽取其中所有nnn.nnn IP地址，使用[0-9]\{3\}\.[0-0\{3\}\。含义是任意数字出现3次，后跟句点，接着是任意数字出现 3次，后跟句点。

```
$ grep '[0-9]\{3\}\.[0-0\[3\}\.' ipfile
```

China-pub.com 下载

8.3 类名

grep允许使用国际字符模式匹配或匹配模式的类名形式。

表8-1 类名及其等价的正则表达式

类	等价的正则表达式	类	等价的正则表达式
[[:upper:]]	[A-Z]	[[:alnum:]]	[0-9a-zA-Z]
[[:lower:]]	[a-z]	[[:space:]]	空格或tab键
[[:digit:]]	[0-9]	[[:alpha:]]	[a-zA-Z]

现举例说明其使用方式。要抽取产品代码，该代码以 5开头，后跟至少两个大写字母。使用的脚本如下：

```
$ grep '5[[:upper:]][[:upper:]]' data.f
483     Sept    5AP1996  USP     65.00    LVX2C  189
483     may     5PA1998  USP     37.00    KVM9D  644
```

如果要抽取以P或D结尾的所有产品代码，方法如下：

```
$ grep '[[:upper:]][[:upper:]] [P,D]' data.f
483     Sept    5AP1996  USP     65.00    LVX2C  189
219     dec     2CC1999  CAD     23.00    PLV2C  68
484     nov     7PL1996  CAD     49.00    PLV2C  234
483     may     5PA1998  USP     37.00    KVM9D  644
216     sept    3ZL1998  USP     86.00    KVM9E  234
```

使用通配符*的匹配模式

现在讲述grep中通配符*的使用。现有文件如下：

```
$ pg testfile
looks
likes
looker
long
```

下述grep模式结果显示如下：

```
$ grep 'l.*s' testfile
looks
likes


$ grep 'l.*k.' testfile
looks
likes


$ grep 'ooo*' testfile
looks
```

如在行尾查询某一单词，试如下模式：

```
$ grep 'device$' *
```

这将在所有文件中查询行尾包含单词 device的所有行。

8.4 系统grep命令

使用已学过的知识可以很容易通过 grep命令获得系统信息。下面几个例子中，将用到管

China-pub.com
下载

道命令，即符号|，使用它左边命令的输出结果作为它右边命令的输入。

8.4.1 目录

如果要查询目录列表中的目录，方法如下：

```
$ ls -l | grep '^d'
```

如果在一个目录中查询不包含目录的所有文件，方法如下：

```
$ ls -l | grep '^[^d]'
```

要查询其他用户和其他用户组成员有可执行权限的目录集合，方法如下：

```
$ ls -l | grep '^d.....x..x'
```

8.4.2 passwd文件

```
$ grep "louise" /etc/passwd
louise:lxAL6GW9G.ZyY:501:501:Accounts Sect 1C:/home/accts/louise:
/bin/sh
```

上述脚本查询/etc/passwd文件是否包含louise字符串，如果误输入以下脚本：

```
$ grep "louise" /etc/password
```

将返回grep命令错误代码'No such file or directory'。

上述结果表明输入文件名不存在，使用grep命令-s开关，可屏蔽错误信息。

```
$ grep -s "louise" /etc/password
$
```

返回命令提示符，而没有文件不存在的错误提示。

如果grep命令不支持-s开关，可替代使用以下命令：

```
$ grep "louise" /etc/password >/dev/null 2>&1
```

脚本含义是匹配命令输出或错误（2>$1），并将结果输出到系统池。大多数系统管理员称/dev/null为比特池，没关系，可以将之看成一个无底洞，有进没有出，永远也不会填满。

上述两个例子并不算好，因为这里的目的只想知道查询是否成功。本书后面部分将讨论grep命令的exit用法，它允许查询并不成功返回。

如要保存grep命令的查询结果，可将命令输出重定向到一个文件。

```
$ grep "louise" /etc/passwd >/tmp/passwd.out
```

脚本将输出重定向到目录/tmp下文件passwd.out中。

8.4.3 使用ps命令

使用带有ps x命令的grep可查询系统上运行的进程。ps x命令意为显示系统上运行的所有进程列表。要查看DNS服务器是否正在运行（通常称为named），方法如下：

```
$ ps ax | grep "named"
PID  TTY  STAT TIME  CMD
211  ?    S    4.56  named
303  3    S    0.00  grep named
```

输出也应包含此grep命令，因为grep命令创建了相应进程，ps x将找到它。在grep命令中使用-v选项可丢弃ps命令中的grep进程。

China-pub.com 下载

```
$ ps ax | grep named |grep -v "grep"
211  ?    S    4.56  named
```

如果ps x不适用于用户系统，替代使用 ps -ef。

8.4.4 对一个字符串使用grep

grep不只应用于文件，也可应用于字符串。为此使用 echo字符串命令，然后对 grep命令使用管道输入。

```
$ STR="Mary Joe Peter Pauline"
$ echo $STR | grep "Mary"
Mary Joe Peter Pauline
```

匹配成功实现。

```
$ echo $STR | grep "Simon"
```

因为没有匹配字符串，所以没有输出结果。

8.5 egrep

egrep代表expression或extended grep，适情况而定。egrep接受所有的正则表达式，egrep的一个显著特性是可以以一个文件作为保存的字符串，然后将之传给 egrep作为参数，为此使用-f开关。如果创建一个名为grepstrings的文件，并输入484和47：

```
$ pg grepstrings
484
47
$ egrep -f grepstrings data.f
```

上述脚本匹配data.f中包含484或47的所有记录。当匹配大量模式时，-f开关很有用，而在一个命令行中敲入这些模式显然极为繁琐。

如果要查询存储代码32L或2CC，可以使用（|）符号，意即“|”符号两边之一或全部。

```
$ egrep '(3ZL|2CC)' data.f
47      Oct     3ZL1998  LPSX    43.00   KVM9D   512
219     dec     2CC1999  CAD     23.00   PLV2C   68
216     sept    3ZL1998  USP     86.00   KVM9E   234
```

可以使用任意多竖线符“|”，例如要查看在系统中是否有帐号 louise、matty或pauline，使用who命令并管道输出至egrep。

```
$ who | egrep (louise|matty|pauline)
 louise pty8
 matty tty02
pauline pty2
```

还可以使用^符号排除字符串。如果要查看系统上的用户，但不包括 matty和pauline，方法如下：

```
$ who | egrep -v '^(matty|pauline)'
```

如果要查询一个文件列表，包括 shutdown、shutdowns、reboot和reboots，使用egrep可容易地实现。

```
$ egrep '(shutdown | reboot) (s)?'  *
```

China-pub.com
下载

8.6 小结

希望大家已经理解了grep的灵活性，它是一个很强大而流行的工具，像其他许多 UNIX工具一样，已经被使用在 DOS中。如果要通过文件快速查找字符串或模式， grep是一个很好的选择。简单地说， grep是shell编程中很重要的工具，在本书后面部分使用其他 UNIX工具和进行变量替换时将发现这一点。

China-pub.com
下载

第9章 AWK 介 绍

如果要格式化报文或从一个大的文本文件中抽取数据包，那么 awk可以完成这些任务。它在文本浏览和数据的熟练使用上性能优异。

整体来说，awk是所有shell过滤工具中最难掌握的，不知道为什么，也许是其复杂的语法或含义不明确的错误提示信息。在学习 awk语言过程中，就会慢慢掌握诸如 Bailing out 和awk:cmd.Line:等错误信息。可以说awk是一种自解释的编程语言，之所以要在 shell中使用awk是因为awk本身是学习的好例子，但结合 awk与其他工具诸如grep和sed，将会使shell编程更加容易。

本章没有讲述awk的全部特性，也不涉及awk的深层次编程，（这些可以在专门讲述awk的书籍中找到）。本章仅注重于讲述使用awk执行行操作及怎样从文本文件和字符串中抽取信息。

本章内容有：

- 抽取域。
- 匹配正则表达式。
- 比较域。
- 向awk传递参数。
- 基本的awk行操作和脚本。

本书几乎所有包含 awk命令的脚本都结合了 sed和grep，以从文本文件和字符串中抽取信息。为获得所需信息，文本必须格式化，意即用域分隔符划分抽取域，分隔符可能是任意字符，在以后讲述awk时再详细讨论。

awk以发展这种语言的人 Aho.Weninberger和Kernigham命名。还有nawk和gawk，它们扩展了文本特性，但本章不予讨论。

awk语言的最基本功能是在文件或字符串中基于指定规则浏览和抽取信息。 awk抽取信息后，才能进行其他文本操作。完整的awk脚本通常用来格式化文本文件中的信息。

9.1 调用awk

有三种方式调用awk，第一种是命令行方式，如：

awk [-F field-separator] 'commands' input-file(s)

这里，commands是真正的awk命令。本章将经常使用这种方法。

上面例子中，[-F域分隔符]是可选的，因为awk使用空格作为缺省的域分隔符，因此如果要浏览域间有空格的文本，不必指定这个选项，但如果要浏览诸如 passwd文件，此文件各域以冒号作为分隔符，则必须指明 -F选项，如：

awk -F: 'commands' input-file

第二种方法是将所有 awk命令插入一个文件，并使 awk程序可执行，然后用 awk命令解释器作为脚本的首行，以便通过键入脚本名称来调用它。

第三种方式是将所有的awk命令插入一个单独文件，然后调用：

awk -f awk-script-file input-files(s)

-f选项指明在文件 awk_script_file中的awk脚本， input_file(s)是使用awk进行浏览的文件名。

9.2 awk脚本

在命令中调用awk时，awk脚本由各种操作和模式组成。

如果设置了-F选项，则awk每次读一条记录或一行，并使用指定的分隔符分隔指定域，但如果未设置-F选项，awk假定空格为域分隔符，并保持这个设置直到发现一新行。当新行出现时，awk命令获悉已读完整条记录，然后在下一个记录启动读命令，这个读进程将持续到文件尾或文件不再存在。

参照表9-1，awk每次在文件中读一行，找到域分隔符（这里是符号 #），设置其为域n，直至一新行（这里是缺省记录分隔符），然后，划分这一行作为一条记录，接着 awk再次启动下一行读进程。

表9-1 awk读文件记录的方式

域1	分 隔 符	域2	分 隔 符	域 3	分 隔 符	域4及换行
P.Bunny(记录1)	#	02/99	#	48	#	Yellow \n
J.Troll(记录2)	#	07/99	#	4842	#	Brown-3 \n

9.2.1 模式和动作

任何awk语句都由模式和动作组成。在一个 awk脚本中可能有许多语句。模式部分决定动作语句何时触发及触发事件。处理即对数据进行的操作。如果省略模式部分，动作将时刻保持执行状态。

模式可以是任何条件语句或复合语句或正则表达式。模式包括两个特殊字段 BEGIN和END。使用BEGIN语句设置计数和打印头。 BEGIN语句使用在任何文本浏览动作之前，之后文本浏览动作依据输入文件开始执行。 END语句用来在awk完成文本浏览动作后打印输出文本总数和结尾状态标志。如果不特别指明模式， awk总是匹配或打印行数。

实际动作在大括号{}内指明。动作大多数用来打印，但是还有些更长的代码诸如 if和循环（looping）语句及循环退出结构。如果不指明采取动作， awk将打印出所有浏览出来的记录。

下面将深入讲解这些模式和动作。

9.2.2 域和记录

awk执行时，其浏览域标记为$1，$2...$n。这种方法称为域标识。使用这些域标识将更容易对域进行进一步处理。

使用$1,$3表示参照第1和第3域，注意这里用逗号做域分隔。如果希望打印一个有 5个域的记录的所有域，不必指明 $1,$2,$3,$4,$5，可使用 $0，意即所有域。Awk浏览时，到达一新行，即假定到达包含域的记录末尾，然后执行新记录下一行的读动作，并重新设置域分隔。注意执行时不要混淆符号 $和shell提示符$，它们是不同的。

为打印一个域或所有域，使用 print命令。这是一个awk动作（动作语法用圆括号括起来）。

China-pub.com 下载

1. 抽取域

真正执行前看几个例子，现有一文本文件 grade.txt，记录了一个称为柔道数据库的行信息。

```
$ pg grade.txt
M.Tansley   05/99    48311   Green     8    40    44
J.Lulu      06/99    48317   green     9    24    26
P.Bunny     02/99    48      Yellow    12   35    28
J.Troll     07/99    4842    Brown-3   12   26    26
L.Tansley   05/99    4712    Brown-2   12   30    28
```

此文本文件有7个域，即（1）名字、（2）升段日期、（3）学生序号、（4）腰带级别、（5）年龄、（6）目前比赛积分、（7）比赛最高分。

因为域间使用空格作为域分隔符，故不必用 -F选项划分域，现浏览文件并导出一些数据。在例子中为了利于显示，将空格加宽使各域看得更清晰。

2. 保存awk输出

有两种方式保存shell提示符下awk脚本的输出。最简单的方式是使用输出重定向符号 >文件名，下面的例子重定向输出到文件 wow。

```
$ awk '{print $0}' grade.txt >wow
```

使用这种方法要注意，显示屏上不会显示输出结果。因为它直接输出到文件。只有在保证输出结果正确时才会使用这种方法。它也会重写硬盘上同名数据。

第二种方法是使用 tee命令，在输出到文件的同时输出到屏幕。在测试输出结果正确与否时多使用这种方法。例如输出重定向到文件 delete_me_and_die，同时输出到屏幕。使用这种方法，在awk命令结尾写入 | tee delete_me_and_die。

```
$ awk '{print $0}' grade.txt | tee delete_me_and_die
```

3. 使用标准输入

在深入讲解这一章之前，先对 awk脚本的输入方法简要介绍一下。实际上任何脚本都是从标准输入中接受输入的。为运行本章脚本，使用 awk脚本输入文件格式，例如：

```
$ belts.awk grade_student.txt
```

也可替代使用下述格式：

使用重定向方法：

```
$ belts.awk < grade2.txt
```

或管道方法：

```
$ grade2.txt|belts.awk
```

4. 打印所有记录

```
$ awk '{print $0}' grade.txt
```

awk读每一条记录。因为没有模式部分，只有动作部分 {print $0}(打印所有记录)，这个动作必须用花括号括起来。上述命令打印整个文件。

```
M.Tansley   05/99    48311   Green     8    40    44
J.Lulu      06/99    48317   green     9    24    26
P.Bunny     02/99    48      Yellow    12   35    28
J.Troll     07/99    4842    Brown-3   12   26    26
L.Tansley   05/99    4712    Brown-2   12   30    28
```

5. 打印单独记录

假定只打印学生名字和腰带级别，通过查看域所在列，可知为 field-1和field-4，因此可以使用$1和$4，但不要忘了加逗号以分隔域。

```
$ awk  '{print $1,$4}' grade.txt
M.Tansley   Green
J.Lulu      green
P.Bunny     Yellow
J.Troll     Brown-3
L.Tansley   Brown-2
```

6. 打印报告头

上述命令输出在名字和腰带级别之间用一些空格使之更容易划分，也可以在域间使用 tab键加以划分。为加入tab键，使用tab键速记引用符 \t，后面将对速记引用加以详细讨论。也可以为输出文本加入信息头。本例中加入 name和belt及下划线。下划线使用 \n，强迫启动新行，并在\n下一行启动打印文本操作。打印信息头放置在 BEGIN模式部分，因为打印信息头被界定为一个动作，必须用大括号括起来。在 awk查看第一条记录前，信息头被打印。

```
$ awk  'BEGIN {print "Name    Belt\n-------------------------------"}
{print $1"\t"$4}' grade.txt


Name                Belt
---------------------------------------
M.Tansley           Green
J.Lulu              green
P.Bunny             Yellow
J.Troll             Brown-3
L.Tansley           Brown-3
```

7. 打印信息尾

如果在末行加入end of report信息，可使用END语句。END语句在所有文本处理动作执行完之后才被执行。 END语句在脚本中的位置放置在主要动作之后。下面简单打印头信息并告之查询动作完成。

```
$ awk 'BEGIN {print "Name\n-------"} {print $1} END {"end-of-report"}'
grade.txt
Name
-------
M.Tansley
J.Lulu
P.Bunny
J.Troll
L.Tansley
end-of-report
```

8. awk错误信息提示

几乎可以肯定，在使用 awk时，将会在命令中碰到一些错误。 awk将试图打印错误行，但由于大部分命令都只在一行，因此帮助不大。

系统给出的显示错误信息提示可读性不好。使用上述例子，如果丢了一个双引号， awk将返回：

```
$ awk 'BEGIN {print "Name\n-------} {print $1} END {"end-of-report"}''
grade.txt

awk: cmd. line:1: BEGIN {print "Name\n-------} {print $1} END
```

China-pub.com 下载

```
{"end-of -report"}
awk: cmd. line:1: ^ unterminated string
```

当第一次使用awk时，可能被错误信息搅得不知所措，但通过长时间和不断的学习，可总结出以下规则。在碰到awk错误时，可相应查找：

- 确保整个awk命令用单引号括起来。
- 确保命令内所有引号成对出现。
- 确保用花括号括起动作语句，用圆括号括起条件语句。
- 可能忘记使用花括号，也许你认为没有必要，但awk不这样认为，将按之解释语法。

如果查询文件不存在，将得到下述错误信息：

```
$ awk 'END {print NR}' grades.txt

awk: cmd. line:2: fatal: cannot open file 'grades.txt' for
reading (No such file or directory)
```

9. awk键盘输入

如果在命令行并没有输入文件grade.txt，将会怎样？

```
$ awk 'BEGIN {print "Name  Belt\n--------------------------------"}
      {print $1"\t"$4}'

Name                Belt
--------------------------------
>
```

BEGIN部分打印了文件头，但awk最终停止操作并等待，并没有返回shell提示符。这是因为awk期望获得键盘输入。因为没有给出输入文件，awk假定下面将会给出。如果愿意，顺序输入相关文本，并在输入完成后敲<Ctrl-D>键。如果敲入了正确的域分隔符，awk会像第一个例子一样正常处理文本。这种处理并不常用，因为它大多应用于大量的打印稿。

9.2.3 awk中正则表达式及其操作

在grep一章中，有许多例子用到正则表达式，这里将不使用同样的例子，但可以使用条件操作讲述awk中正则表达式的用法。

这里正则表达式用斜线括起来。例如，在文本文件中查询字符串Green，使用/Green/可以查出单词Green的出现情况。

9.2.4 元字符

这里是awk中正则表达式匹配操作中经常用到的字符，详细情况请参阅本书第7章正则表达式概述。

```
\ ^ $ . [] | () * + ?
```

这里有两个字符第7章没有讲到，因为它们只适用于awk而不适用于grep或sed。它们是：

\+ 使用+匹配一个或多个字符。

？ 匹配模式出现频率。例如使用/XY?Z/匹配XYZ或YZ。

9.2.5 条件操作符

表9-2给出awk条件操作符，后面将给出其用法。

China-pub.com
下载

表9-2 awk条件操作符

操 作 符	描 述	操 作 符	描 述
<	小于	>=	大于等于
<=	小于等于	~	匹配正则表达式
==	等于	!~	不匹配正则表达式
!=	不等于		

1. 匹配

为使一域号匹配正则表达式，使用符号‘～’后紧跟正则表达式，也可以用 if语句。awk中if后面的条件用（）括起来。

观察文件grade.txt，如果只要打印brown腰带级别可知其所在域为field-4，这样可以写出表达式{if($4~/brown/) print }意即如果field-4包含brown，打印它。如果条件满足，则打印匹配记录行。可以编写下面脚本，因为这是一个动作，必须用花括号{}括起来。

```
$ awk  {if($4~/Brown/) print $0}' grade.txt
J.Troll      07/99   4842     Brown-3    12     26       26
L.Tansley    05/99   4712     Brown-2    12     30       28
```

匹配记录找到时，如果不特别声明， awk缺省打印整条记录。使用if语句开始有点难，但不要着急，因为有许多方法可以跳过它，并仍保持同样结果。下面例子意即如果记录包含模式brown，就打印它：

```
$ awk  '$0 ~ /Brown/' grade.txt
J.Troll      07/99     4842    Brown-3  12     26       26
L.Tansley    05/99     4712    Brown-2  12     30       28
```

2. 精确匹配

假定要使字符串精确匹配，比如说查看学生序号 48，文件中有许多学生序号包含 48，如果在field-3中查询序号48，awk将返回所有序号带48的记录：

```
$ awk  '{if($3~/48/) print $0}' grade.txt
M.Tansley    05/99     48311   Green    8      40    44
J.Lulu       06/99     48317   green    9      24    26
P.Bunny      02/99     48      Yellow   12     35    28
J.Troll      07/99     4842    Brown-3  12     26    26
```

为精确匹配48，使用等号==，并用单引号括起条件。例如 $3== “ 48 ”，这样确保只有48序号得以匹配，其余则不行。

```
$ awk '$3=="48" {print $0}' grade.txt
P.Bunny      02/99    48       Yellow   12     35    28
```

3. 不匹配

有时要浏览信息并抽取不匹配操作的记录，与 ~相反的符号是!~，意即不匹配。像原来使用查询brown腰带级别的匹配操作一样，现在看看不匹配情况。表达式 $0 !~/brown/，意即查询不包含模式brown腰带级别的记录并打印它。

注意，缺省情况下，awk将打印所有匹配记录，因此这里不必加入动作部分。

```
$ awk  '$0 !~ /Brown/' grade.txt
M.Tansley    05/99     48311   Green    8     40     44
J.Lulu       06/99     48317   green    9     24     26
P.Bunny      02/99     48      Yellow   12    35     28
```

China-pub.com 下载

可以只对field-4进行不匹配操作，方法如下：

```
$ awk '{if($4!~/Brown/) print $0}' grade.txt
M.Tansley    05/99    48311  Green    8    40   44
J.Lulu       06/99    48317  green    9    24   26
P.Bunny      02/99    48     Yellow   12   35   28
```

如果只使用命令awk$4 !="brown"{print $0} grade.txt，将返回错误结果，因为用引号括起了brown，将只匹配‘brown而不匹配brown-2和brown-3，当然，如果想要查询非brown-2的腰带级别，可做如下操作：

```
$ awk '$4 != "Brown-2" {print $0}' grade.txt
```

4. 小于

看看哪些学生可以获得升段机会。测试这一点即判断目前级别分 field-6是否小于最高分field-7，在输出结果中，加入这一改动很容易。

```
$ awk '{if ($6 < $7) print $0 "$1 Try better at the next comp"}'
grade.txt
M.Tansley Try better at the next comp
J.Lulu Try better at the next comp
```

5. 小于等于

对比小于，小于等于只在操作符上做些小改动，满足此条件的记录也包括上面例子中的输出情况。

```
$ awk '{if ($6 <= $7) print $1}' grade.txt
M.Tansley
J.Lulu
J.Troll
```

6. 大于

大于符号大家都熟知，请看例子：

```
$ awk '{if ($6 > $7) print $1}' grade.txt
L.Tansley
P.Bunny
```

希望读者已经掌握了操作符的基本用法。

7. 设置大小写

为查询大小写信息，可使用[]符号。在测试正则表达式时提到可匹配[]内任意字符或单词，因此若查询文件中级别为green的所有记录，不论其大小写，表达式应为‘/[Gg]reen/：’

```
$ awk '/[Gg]reen/' grade.txt
M.Tansley    05/99    48311  Green    8    40   44
J.Lulu       06/99    48317  green    9    24   26
```

8. 任意字符

抽取名字，其记录第一域的第四个字符是 a，使用句点.。表达式/^...a/意为行首前三个字符任意，第四个是a，尖角符号代表行首。

```
$ awk '$1 ~/^ . . . a/' grade.txt
M.Tansley    05/99    48311  Green    8    40   44
L.Tansley    05/99    4712   Brown-2  12   30   28
```

9. 或关系匹配

为抽取级别为yellow或brown的记录，使用竖线符 |。意为匹配 | 两边模式之一。注意，使用竖线符时，语句必须用圆括号括起来。

China-pub.com 下载


```
$ awk '$0~/(Yellow|Brown)/' grade.txt
P.Bunny     02/99    48      Yellow   12   35    28
J.Troll     07/99    4842    Brown-3  12   26    26
L.Tansley   05/99    4712    Brown-2  12   30    28
```

上面例子输出所有级别为Yellow或Brown的记录。

使用这种方法在查询级别为Green或green时，可以得到与使用[]表达式相同的结果。

```
$ awk '$0~/(Green|green)/' grade.txt
M.Tansley   05/99    48311   Green    8     40   44
J.Lulu      06/99    48317   green    9     24   26
```

10. 行首

不必总是使用域号。如果查询文本文件行首包含48的代码，可简单使用下面^符号：

```
$ awk '/^48/' input-file
```

这里讲述了在awk中怎样使用第7章中涉及的表达式。像第7章的开头提到的，所有表达式（除字符重复出现外）在awk中都是合法的。

复合模式或复合操作符用于形成复杂的逻辑操作，复杂程度取决于编程者本人。有必要了解的是，复合表达式即为模式间通过使用下述各表达式互相结合起来的表达式：

&& AND：语句两边必须同时匹配为真。

|| OR：语句两边同时或其中一边匹配为真。

! 非 求逆

11. AND

打印记录，使其名字为‘ P.Bunny且级别为Yellow，使用表达式($1=="P.Bunny"&& $4=="Yellow")，意为&&两边匹配均为真。完整命令如下：

```
$ awk '{if ($1=="P.Bunny" && $4=="Yellow")print $0}' grade.txt
P.Bunny     02/99     48     Yellow   12    35   28
```

12. Or

如果查询级别为Yellow或Brown，使用或命令。意为“ || ”符号两边的匹配模式之一或全部为真。

```
$ awk '{if ($4=="Yellow" || $4~/Brown/) print $0}' grade.txt
P.Bunny     02/99     48      Yellow   12    35    28
J.Troll     07/99     4842    Brown-3  12    26    26
L.Tansley   05/99     4712    Brown-2  12    30    28
```

9.2.6 awk内置变量

awk有许多内置变量用来设置环境信息。这些变量可以被改变。表9-3显示了最常使用的一些变量，并给出其基本含义。

表9-3 awk内置变量

ARGC	命令行参数个数
ARGV	命令行参数排列
ENVIRON	支持队列中系统环境变量的使用
FILENAME	awk浏览的文件名
FNR	浏览文件的记录数
FS	设置输入域分隔符，等价于命令行 -F选项

China-pub.com 下载

(续)

NF	浏览记录的域个数
NR	已读的记录数
OFS	输出域分隔符
ORS	输出记录分隔符
RS	控制记录分隔符

ARGC支持命令行中传入awk脚本的参数个数。ARGV是ARGC的参数排列数组，其中每一元素表示为ARGV[n]，n为期望访问的命令行参数。

ENVIRON支持系统设置的环境变量，要访问单独变量，使用实际变量名，例如ENVIRON[“EDITOR”]=“Vi”。

FILENAME支持awk脚本实际操作的输入文件。因为awk可以同时处理许多文件，因此如果访问了这个变量，将告之系统目前正在浏览的实际文件。

FNR支持awk目前操作的记录数。其变量值小于等于NR。如果脚本正在访问许多文件，每一新输入文件都将重新设置此变量。

FS用来在awk中设置域分隔符，与命令行中-F选项功能相同。缺省情况下为空格。如果用逗号来作域分隔符，设置FS=","。

NF支持记录域个数，在记录被读之后再设置。

OFS允许指定输出域分隔符，缺省为空格。如果想设置为#，写入OFS="#"。

ORS为输出记录分隔符，缺省为新行（\n)。

RS是记录分隔符，缺省为新行(\n)。

9.2.7 NF、NR和FILENAME

下面看一看awk内置变量的例子。

要快速查看记录个数，应使用NR。比如说导出一个数据库文件后，如果想快速浏览记录个数，以便对比于其初始状态，查出导出过程中出现的错误。使用NR将打印输入文件的记录个数。print NR放在END语法中。

```
$ awk 'END {print NR}' grade.txt
```

以下例子中，所有学生记录被打印，并带有其记录号。使用NF变量显示每一条读记录中有多少个域，并在END部分打印输入文件名。

```
$ awk '{print NF,NR,$0}END{print FILENAME}' grade.txt
7  1 M.Tansley  05/99   48311   Green     8       40       44
7  2 J.Lulu     06/99   48317   green     9       24       26
7  3 P.Bunny    02/99   48      Yellow    12      35       28
7  4 J.Troll    07/99   4842    Brown-3   12      26       26
7  5 L.Tansley  05/99   4712    Brown-2   12      30       28
grade.txt
```

在从文件中抽取信息时，最好首先检查文件中是否有记录。下面的例子只有在文件中至少有一个记录时才查询Brown级别记录。使用AND复合语句实现这一功能。意即至少存在一个记录后，查询字符串Brown，最后打印结果。

```
$ awk '{if (NR >0 && $4~/Brown/)print $0}' grade.txt
J.Troll       07/99     4842     Brown-3   12    26       26
L.Tansley     05/99     4712     Brown-2   12    30       28
```

NF的一个强大功能是将变量$PWD的返回值传入awk并显示其目录。这里需要指定域分隔符/。

```
$ pwd
/usr/local/etc
$ echo $PWD | awk -F/ '{print $NF}'
etc
```

另一个例子是显示文件名。

```
$ echo "/usr/local/etc/rc.sybase" | awk -F/ '{print $NF}'
rc.sybase
```

9.2.8 awk操作符

在awk中使用操作符，基本表达式可以划分为数字型、字符串型、变量型、域及数组元素，前面已经讲过一些。下面列出其完整列表。

在表达式中可以使用下述任何一种操作符。

= += *= / = %= ^ =	赋值操作符
?	条件表达操作符
|| && !	并、与、非（上一节已讲到）
~!~	匹配操作符，包括匹配和不匹配
< <= == != >>	关系操作符
+ - * / % ^	算术操作符
+ + --	前缀和后缀

前面已经讲到了其中几种操作，下面继续讲述未涉及的部分。

1. 设置输入域到域变量名

在awk中，设置有意义的域名是一种好习惯，在进行模式匹配或关系操作时更容易理解。一般的变量名设置方式为name=$n，这里name为调用的域变量名，n为实际域号。例如设置学生域名为name，级别域名为belt，操作为name=$1;belts=$4。注意分号的使用，它分隔awk命令。下面例子中，重新赋值学生名域为 name，级别域为belts。查询级别为Yellow的记录，并最终打印名称和级别。

```
$ awk '{name=$1;belts=$4; if(belts ~/Yellow/)print name" is belt
"belts}' grade.txt
P.Bunny is belt Yellow
```

2. 域值比较操作

有两种方式测试一数值域是否小于另一数值域。

1) 在BEGIN中给变量名赋值。

2) 在关系操作中使用实际数值。

通常在BEGIN部分赋值是很有益的，可以在awk表达式进行改动时减少很多麻烦。

使用关系操作必须用圆括号括起来。

下面的例子查询所有比赛中得分在27点以下的学生。

用引号将数字引用起来是可选的，“27”、27产生同样的结果。

```
$ awk '{if($6 < 27)print$0}' grade.txt
J.Lulu      06/99   48317   green    9     24      26
J.Troll     07/99   4842    Brown-3  12    26      26
```

China-pub.com 下载

第二个例子中给数字赋以变量名BASELINE和在BEGIN部分给变量赋值，两者意义相同。

```
$ awk 'BEGIN {BASELINE="27  "}{if($6 < BASELINE)print$0}'  grade.txt
J.Lulu      06/99   48317   green    9     24      26
J.Troll     07/99   4842    Brown-3 12     26      26
```

3. 修改数值域取值

当在awk中修改任何域时，重要的一点是要记住实际输入文件是不可修改的，修改的只是保存在缓存里的awk复本。awk会在变量NR或NF变量中反映出修改痕迹。

为修改数值域，简单的给域标识重赋新值，如：\$1=\$1+5，会将域1数值加5，但要确保赋值域其子集为数值型。

修改M.Tansley的目前级别分域，使其数值从 40减为39，使用赋值语句\$6=\$6-1，当然在实施修改前首先要匹配域名。

```
$ awk '{if($1=="M.Tansley") $6=$6-1; print $1, $6, $7}' grade.txt
M.Tansley   39  44
J.Lulu      24  26
P.Bunny     35  28
J.Troll     26  26
L.Tansley   30  28
```

4. 修改文本域

修改文本域即对其重新赋值。需要做的就是赋给一个新的字符串。在 J.Troll中加入字母，使其成为J.L.Troll，表达式为\$1="J.L.Troll"，记住字符串要使用双秒号（""），并用圆括号括起整个语法。

```
$ awk '{if($1=="J.Troll") ($1="J.L.Troll"); print $1}' grade.txt
M.Tansley
J.Lulu
P.Bunny
J.L.Troll
L.Tansley
```

5. 只显示修改记录

上述例子均是对一个小文件的域进行修改，因此打印出所有记录查看修改部分不成问题，但如果文件很大，记录甚至超过 100，打印所有记录只为查看修改部分显然不合情理。在模式后面使用花括号将只打印修改部分。取得模式，再根据模式结果实施操作，可能有些抽象，现举一例，只打印修改部分。注意花括号的位置。

```
$ awk '{if($1=="J.Troll") {$1="J.L.Troll";print $1}}' grade.txt
J.L.Troll
```

6. 创建新的输出域

在awk中处理数据时，基于各域进行计算时创建新域是一种好习惯。创建新域要通过其他域赋予新域标识符。如创建一个基于其他域的加法新域{\$4=\$2+\$3}，这里假定记录包含3个域，则域4为新建域，保存域2和域3相加结果。

在文件grade.txt中创建新域 8保存域目前级别分与域最高级别分的减法值。表达式为‘{\$8=\$7-\$6}’，语法首先测试域目前级别分小于域最高级别分。新域因此只打印其值大于零的学生名称及其新域值。在BEGIN部分加入tab键以对齐报告头。

```
$ awk 'BEGIN{ print "Name\t Difference"}{if($6 <$7) {$8=$7--$6; print
$1,$8}}' grade.txt
Name        Difference
```

```
M.Tansley    4
J.Lulu       2
```

当然可以创建新域，并赋给其更有意义的变量名。例如：

```
$ awk 'BEGIN{ print "Name\t Difference"}{if($6 <$7) {diff=$7-$6; print
$1,diff}}' grade.txt
M.Tansley    4
J.Lulu       2
```

7. 增加列值

为增加列数或进行运行结果统计，使用符号 +=。增加的结果赋给符号左边变量值，增加到变量的域在符号右边。例如将 $1加入变量total，表达式为total+=$1。列值增加很有用。许多文件都要求统计总数，但输出其统计结果十分繁琐。在 awk中这很简单，请看下面的例子。

将所有学生的‘目前级别分’加在一起，方法是 tot+=$6，tot即为awk浏览的整个文件的域6结果总和。所有记录读完后，在 END部分加入一些提示信息及域 6总和。不必在 awk中显示说明打印所有记录，每一个操作匹配时，这是缺省动作。

```
$ awk '(tot+=$6); END{print "Club student total points :" tot}'
grade.txt
M.Tansley    05/99    48311   Green     8     40    44
J.Lulu       06/99    48317   green     9     24    26
P.Bunny      02/99    48      Yellow    12    35    28
J.Troll      07/99    4842    Brown-3   12    26    26
L.Tansley    05/99    4712    Brown-2   12    30    28
Club student total points :155
```

如果文件很大，你只想打印结果部分而不是所有记录，在语句的外面加上圆括号（）即可。

```
$ awk '{(tot+=$6)}; END{print "Club student total points :" tot}'
grade.txt
Club student total points :155
```

8. 文件长度相加

在目录中查看文件时，如果想快速查看所有文件的长度及其总和，但要排除子目录，使用ls -l命令，然后管道输出到awk，awk首先剔除首字符为d（使用正则表达式）的记录，然后将文件长度列相加，并输出每一文件长度及在 END部分输出所有文件的长度。

本例中，首先用ls -l命令查看一下文件属性。注意第二个文件属性首字符为 d，说明它是一个目录，文件长度是第 5列，文件名是第 9列。如果系统不是这样排列文件名及其长度，应适时加以改变。

```
          -rw-r--r--   1  root   root    80    Apr  11  18:56  acc.txt
          drwx------   2  root   root    1024  Mar  26  20:53  nsmail
Columns   1            2  3      4       5     6    7   8      9
```

下面的正则表达式表明必须匹配行首，并排除字符 d，表达式为^[^d]。

使用此模式打印文件名及其长度，然后将各长度相加放入变量 tot中。

```
$ ls -l | awk ' /^[^d]/ {print $9"\t"$5} {tot+=$5} END
{print "total KB:"tot}'
dev_pkg.fail     345
failedlogin      12416
messages         4260
sulog            12810
```

China-pub.com 下载

```
utmp            1856
wtmp            7104
total KB:41351
```

9.2.9 内置的字符串函数

awk有许多强大的字符串函数，见表9-4。

表9-4 awk内置字符串函数

gsub(r,s)	在整个$0中用s替代r
gsub(r,s,t)	在整个t中用s替代r
index(s,t)	返回s中字符串t的第一位置
length(s)	返回s长度
match(s,r)	测试s是否包含匹配r的字符串
split(s,a,fs)	在fs上将s分成序列a
sprint(fmt,exp)	返回经fmt格式化后的exp
sub(r,s)	用$0中最左边最长的子串代替s
substr(s,p)	返回字符串s中从p开始的后缀部分
substr(s,p,n)	返回字符串s中从p开始长度为n的后缀部分

gsub函数有点类似于sed查找和替换。它允许替换一个字符串或字符为另一个字符串或字符，并以正则表达式的形式执行。第一个函数作用于记录 $0，第二个gsub函数允许指定目标，然而，如果未指定目标，缺省为$0。

index（s，t）函数返回目标字符串s中查询字符串t的首位置。length函数返回字符串s字符长度。match函数测试字符串s是否包含一个正则表达式 r定义的匹配。split使用域分隔符fs将字符串s划分为指定序列a。sprint函数类似于printf函数（以后涉及），返回基本输出格式fmt的结果字符串exp。sub（r，s）函数将用s替代$0中最左边最长的子串，该子串被（ r）匹配。sub（s，p）返回字符串s在位置p后的后缀。substr（s，p，n）同上，并指定子串长度为n。

现在看一看awk中这些字符串函数的功能。

1. gsub

要在整个记录中替换一个字符串为另一个，使用正则表达式格式， /目标模式/，替换模式/。例如改变学生序号4842到4899：

```
$ awk 'gsub(/4842/,4899) {print $0}' grade.txt
J.Troll     07/99     4899    Brown-3  12    26    26
```

2. index

查询字符串s中t出现的第一位置。必须用双引号将字符串括起来。例如返回目标字符串Bunny中ny出现的第一位置，即字符个数。

```
$ awk 'BEGIN {print index("Bunny","ny")}' grade.txt
4
```

3. length

返回所需字符串长度，例如检验字符串 J.Troll返回名字及其长度，即人名构成的字符个数。

```
$ awk '$1=="J.Troll" {print length($1)" "$1}' grade.txt
7 J.Troll
```

还有一种方法，这里字符串加双引号。

China-pub.com 下载


```
$ awk 'BEGIN {print length("A FEW GOOD MEN")}'
14
```

4. match

match测试目标字符串是否包含查找字符的一部分。可以对查找部分使用正则表达式，返回值为成功出现的字符排列数。如果未找到，返回 0，第一个例子在ANCD中查找d。因其不存在，所以返回0。第二个例子在ANCD中查找D。因其存在，所以返回ANCD中D出现的首位置字符数。第三个例子在学生J.Lulu中查找u。

```
$ awk 'BEGIN {print match("ANCD",/d/)}'
0
$ awk 'BEGIN {print match("ANCD",/C/)}'
3
$ awk '$1=="J.Lulu" {print match($1,"u")}' grade.txt
4
```

5. split

使用split返回字符串数组元素个数。工作方式如下：如果有一字符串，包含一指定分隔符 -，例如AD2-KP9-JU2-LP-1，将之划分成一个数组。使用 split，指定分隔符及数组名。此例中，命令格式为("AD2-KP9-JU2-LP-1"，parts_array，"-")，split然后返回数组下标数，这里结果为4。

还有一个例子使用不同的分隔符。

```
$ awk 'BEGIN {print split("123#456#678", myarray, "#")}'
3
```

这个例子中，split返回数组myarray的下标数。数组myarray取值如下：

Myarray[1]="123"
Myarray[2]="456"
Myarray[3]="678"

本章结尾部分讲述数组概念。

6. sub

使用sub发现并替换模式的第一次出现位置。字符串 STR包含‘poped popo pill’，执行下列sub命令sub（/op/，"op"，STR）。模式op第一次出现时，进行替换操作，返回结果如下：‘pOPed pope pill’。

本章文本文件中，学生J.Troll的记录有两个值一样，“目前级别分”与“最高级别分”。只改变第一个为29，第二个仍为24不动，操作命令为sub（/26/，"29"，$0），只替换第一个出现24的位置。注意J.Troll记录需存在。

```
$ awk '$1=="J.Troll" sub(/26/,"29",$0)' grade.txt
M.Tansley    05/99    48311  Green    8    40   44
J.Lulu       06/99    48317  green    9    24   29
P.Bunny      02/99    48     Yellow   12   35   28
J.Troll      07/99    4842   Brown-3  12   29   26
L.Tansley    05/99    4712   Brown-2  12   30   28
```

7. substr

substr是一个很有用的函数。它按照起始位置及长度返回字符串的一部分。例子如下：

```
$ awk '$1=="L.Tansley" {print substr($1,1,5)}' grade.txt
L.Tan
```

上面例子中，指定在域1的第一个字符开始，返回其前面5个字符。

China-pub.com 下载

如果给定长度值远大于字符串长度， awk将从起始位置返回所有字符，要抽取 L Tansl-ey的姓，只需从第3个字符开始返回长度为7。可以输入长度99，awk返回结果相同。

```
$ awk '$1=="L.Tansley" {print substr($1,3,99)}' grade.txt
Tansley
```

substr的另一种形式是返回字符串后缀或指定位置后面字符。这里需要给出指定字符串及其返回字串的起始位置。例如，从文本文件中抽取姓氏，需操作域 1，并从第三个字符开始：

```
$ awk '{print substr($1,3)}' grade.txt
Tansley
Lulu
Bunny
Troll
Tansley
```

还有一个例子，在 BEGIN部分定义字符串，在 END部分返回从第 t个字符开始抽取的子串。

```
$ awk 'BEGIN {STR="A FEW GOOD MEN"}END{print substr(STR,7)}' grade.txt
GOOD MEN
```

8. 从shell中向awk传入字符串

本章开始已经提到过， awk脚本大多只有一行，其中很少是字符串表示的。本书大多要求在一行内完成awk脚本，这一点通过将变量传入 awk命令行会变得很容易。现就其基本原理讲述一些例子。

使用管道将字符串stand-by传入awk，返回其长度。

```
$ echo "Stand-by" |awk '{print length($0)}'
8
```

设置文件名为一变量，管道输出到 awk，返回不带扩展名的文件名。

```
$ STR="mydoc.txt"
$ echo $STR| awk '{print substr($STR,1,5)}'
mydoc
```

设置文件名为一变量，管道输出到 awk，只返回其扩展名。

```
$ STR="mydoc.txt"
$ echo $STR| awk '{print substr($STR,7)}'
txt
```

9.2.10 字符串屏蔽序列

使用字符串或正则表达式时，有时需要在输出中加入一新行或查询一元字符。

打印一新行时，（新行为字符 \n），给出其屏蔽序列，以不失其特殊含义，用法为在字符串前加入反斜线。例如使用 \n强迫打印一新行。

如果使用正则表达式，查询花括号（{}），在字符前加反斜线，如 /\{/，将在awk中失掉其特殊含义。

表9-5列出awk识别的另外一些屏蔽序列

表9-5 awk中使用的屏蔽序列

\b	退格键	\t	tab键
\f	走纸换页	\ddd	八进制值
\n	新行	\c	任意其他特殊字符，例如\\为反斜线符号
\r	回车键		

使用上述符号，打印May Day，中间夹tab键，后跟两个新行，再打印May Day，但这次使用八进制数104、141、171、分别代表D、a、y。

```
$ awk 'BEGIN {print"\n\May\tDay\n\nMay \104\141\171"}'
May      Day

May      Day
```

注意，\104为D的八进制ASCII码，\141为a的八进制ASCII码，等等。

9.2.11 awk输出函数printf

目前为止，所有例子的输出都是直接到屏幕，除了 tab键以外没有任何格式。 awk提供函数printf，拥有几种不同的格式化输出功能。例如按列输出、左对齐或右对齐方式。

每一种printf函数（格式控制字符）都以一个 %符号开始，以一个决定转换的字符结束。转换包含三种修饰符。

printf函数基本语法是printf（[格式控制符]，参数），格式控制字符通常在引号里。

9.2.12 printf修饰符

表9-6 awk printf修饰符

-	左对齐
Width	域的步长，用0表示0步长
.prec	最大字符串长度，或小数点右边的位数

表9-7 awk printf格式

%c	ASCII字符
%d	整数
%e	浮点数，科学记数法
%f	浮点数，例如（123.44）
%g	awk决定使用哪种浮点数转换e或者f
%o	八进制数
%s	字符串
%x	十六进制数

1. 字符转换

观察ASCII码中65的等价值。管道输出65到awk。printf进行ASCII码字符转换。这里也加入换行，因为缺省情况下printf不做换行动作。

```
$ echo "65" | awk '{printf "%c\n",$0}'
A
```

当然也可以按同样方式使用awk得到同样结果。

```
$ awk 'BEGIN {printf "%c\n",65}'
A
```

所有的字符转换都是一样的，下面的例子表示进行浮点数转换后‘ 999 ’的输出结果。整数传入后被加了六个小数点。

```
$ awk 'BEGIN {printf "%f\n",999}
999.000000
```

2. 格式化输出

China-pub.com 下载

打印所有的学生名字和序列号，要求名字左对齐， 15个字符长度，后跟序列号。注意 \n 换行符放在最后一个指示符后面。输出将自动分成两列。

```
$ awk {printf "%-15s %s\n", $1,$3}' grade.txt
M.Tansley       48311
J.Lulu          48317
P.Bunny         48
J.Troll         4842
L.Tansley       4712
```

最好加入一些文本注释帮助理解报文含义。可在正文前嵌入头信息。注意这里使用 print 加入头信息。如果愿意，也可使用 printf。

```
$ awk 'BEGIN {print "Name \t\tS.Number"}{printf "%-15s %s\n", $1,$3}'
grade.txt
```

3. 向一行awk命令传值

在查看awk脚本前，先来查看怎样在awk命令行中传递变量。

在awk执行前将值传入awk变量，需要将变量放在命令行中，格式如下：

awk 命令变量=输入文件值

（后面会讲到怎样传递变量到awk脚本中）。

下面的例子在命令行中设置变量 AGE等于10，然后传入awk中，查询年龄在10岁以下的所有学生。

```
$ awk '{if ($5 < AGE) print $0}' AGE=10 grade.txt
M.Tansley   05/99    48311  Green    8    40   44
J.Lulu      06/99    48317  green    9    24   26
```

要快速查看文件系统空间容量，观察其是否达到一定水平，可使用下面 awk一行脚本。因为要监视的已使用空间容量不断在变化，可以在命令行指定一个触发值。首先用管道命令将 df -k 传入awk，然后抽出第 4列，即剩余可利用空间容量。使用 $4~/^[0-9]/取得容量数值（1024块）而不是df的文件头，然后对命令行与‘ if($4<TRIGGER)’上变量TRIGGER中指定的值进行查询测试。

```
$ df -k | awk '($4 ~/^[0-9]/) {if($4 < TRIGGER) print $6"\t"$4}'
TRIGGER=56000
/dos     55808
/apps    51022
```

在系统中使用df -k命令，产生下列信息：

	Filesystem	1024-blocks	Used	Free	%Used	Mounted on
Column	1	2	3	4	5	6

如果系统中df输出格式不同，必须相应改变列号以适应工作系统。

当然可以使用管道将值传入 awk。本例使用 who命令， who命令第一列包含注册用户名，这里打印注册用户，并加入一定信息。

```
$ who | awk '{print $1 " is logged on"}'
louisel is logged on
papam is logged on
```

awk也允许传入环境变量。下面的例子使用环境变量 LOGNAME支持当前用户名。可从who命令管道输出到awk中获得相应信息。

```
$ who | awk '{if ($1 == user) print $1" you are connected to
"$2}'user=$LOGNAME
```

如果root为当前登录用户，输出如下：

```
root you are connected to ttyp1
```

4. awk脚本文件

可以将awk脚本写入一个文件再执行它。命令不必很长（尽管这是写入一个脚本文件的主要原因），甚至可以接受一行命令。这样可以保存 awk命令，以使不必每次使用时都需要重新输入。使用文件的另一个好处是可以增加注释，以便于理解脚本的真正用途和功能。

使用前面的几个例子，将之转换成 awk可执行文件。像原来做的一样，将学生目前级别分相加awk ‘（tot+=$6） END{print "club student total points ："tot} ’ grade.txt。

创建新文件student_tot.awk，给所有awk程序加入awk扩展名是一种好习惯，这样通过查看文件名就知道这是一个awk程序。文本如下：

```
!/bin/awk -f
# all comment lines must start with a hash '#'
# name: student_tot.awk
# to call: student_tot.awk grade.txt
# prints total and average of club student points


#print a header first
BEGIN{
print "Student    Date   Member No.  Grade  Age    Points    Max"
print "Name       Joined                            Gained    Point Available"
print "==================================================================="


}
# let's add the scores of points gained
(tot+=$6)

# finished processing now let's print the total and average point
END{print "Club student total points  :" tot
  print "Average Club Student points:" tot/NR}
```

第一行是！/bin/awk -f。这很重要，没有它自包含脚本将不能执行。这一行告之脚本系统中awk的位置。通过将命令分开，脚本可读性提高，还可以在命令之间加入注释。这里加入头信息和结尾的平均值。基本上这是一个一行脚本文件。

执行时，在脚本文件后键入输入文件名，但是首先要对脚本文件加入可执行权限。

```
$ chmod u+x student_tot.awk
$ student_tot.awk grade.txt
Student     Date      Member No. Grade   Age   Points    Max
Name        Joined                             Gained    Point Available
===================================================================
M.Tansley    05/99     48311   Green     8      40     44
J.Lulu       06/99     48317   green     9      24     26
P.Bunny      02/99     48      Yellow    12     35     28
J.Troll      07/99     4842    Brown-3   12     26     26
L.Tansley    05/99     4712    Brown-2   12     30     28
Club student total points  :155
Average Club Student points:31
```

系统中运用的帐号核实程序检验数据操作人的数据输入，不幸的是这个程序有一点错误，或者应该说是“ 非文本特征 ”。如果一个记录被发现包含一个错误，它应该一次只打印一行

China-pub.com 下载

“ ERROR* ”，但实际上打印了许多这样的错误行。这会给帐号管理员造成误解，因此需要用awk脚本过滤出错误行的出现频率，使得每一个失败记录只对应一个错误行。

在awk实施过滤前先看看部分文件：

```
...
...
INVALID LCSD 98GJ23
ERROR*
ERROR*
CAUTION LPSS ERROR ON ACC NO.
ERROR*
ERROR*
ERROR*
ERROR*
ERROR*
PASS FIELD INVALID ON LDPS
ERROR*
ERROR*
PASS FIELD INVALID ON GHSI
ERROR*
CAUTION LPSS ERROR ON ACC NO.
ERROR*
ERROR*
```

awk脚本如下：

```
#!/bin/awk -f
# error_strip.awk
# to call: error_strip.awk <filename>
# strips out the ERROR* lines if there are more than one
# ERROR* lines after each failed record.

BEGIN { error_line="" }
# tell awk the whole is "ERROR*"
{ if ($0 == "ERROR*" && error_line == "ERROR*")

# go to next line
next;
  error_line = $0; print }
```

awk过滤结果如下：

```
$ strip.awk strip
INVALID LCSD 98GJ23
ERROR*
CAUTION LPSS ERROR ON ACC NO.
ERROR*
PASS FIELD INVALID ON LDPS
ERROR*
PASS FIELD INVALID ON GHSI
ERROR*
CAUTION LPSS ERROR ON ACC NO.
ERROR*
```

5. 在awk中使用FS变量

如果使用非空格符做域分隔符（FS）浏览文件，例如# 或 :，编写这样的一行命令很容易，因为使用FS选项可以在命令行中指定域分隔符。

```
$ awk -F: 'awk {print $0}' input-file
```

China-pub.com
下载

使用awk脚本时，记住设置FS变量是在BEGIN部分。如果不这样做， awk将会发生混淆，不知道域分隔符是什么。

下述脚本指定FS变量。脚本从/etc/passwd文件中抽取第1和第5域，通过分号“；”分隔passwd文件域。第1域是帐号名，第5域是帐号所有者。

```
$ pg passwd.awk
!/bin/awk -f
# to call: passwd.awk /etc/passwd
# print out the first and fifth fields
BEGIN{
FS=":"}
{print $1,"\t",$5}


$ passwd.awk /etc/passwd
root      Special Admin login
xdm       Restart xdm Login
sysadm    Regular Admin login
daemon    Daemon Login for daemons needing permissions
```

6. 向awk脚本传值

向awk脚本传值与向awk一行命令传值方式大体相同，格式为：

```
awk script_file var=value input_file
```

下述脚本对比检查文件中域号和指定数字。这里使用了NF变量MAX，表示指定检查的域号，使用双引号将域分隔符括起来，即使它是一个空格。

```
$ pg fieldcheck.awk
#!/bin/awk -f
# check on how many fields in a file
# name:fieldcheck.awk
# to call: fieldcheck MAX=n FS=<separator> filename
#
NF!=MAX{
print("line " NR " does not have" MAX " fields")}
```

如果以/etc/passwd作输入文件（passwd文件有7个域），运行上述脚本。参数格式如下：

```
$ fieldcheck.awk MAX=7 FS=":" /etc/passwd
```

使用前面一行脚本的例子，将之转换成awk脚本如下：

```
$ pg name.awk
#!/bin/awk -f
# name: age.awk
# to call: age.awk AGE=n grade.txt
# prints ages that are lower than the age supplied on the comand line
{if ($5 < AGE)
  print $0}
```

文本包括了比实际命令更多的信息，没关系，仔细研读文本后，就可以精确知道其功能及如何调用它。

不要忘了增加脚本的可执行权限，然后将变量和赋值放在命令行脚本名字后、输入文件前执行。

```
$ age.awk AGE=10 grade.txt
M.Tansley    05/99     48311   Green     8      40    44
J.Lulu       06/99     48317   green     9      24    26
```

China-pub.com 下载

同样可以使用前面提到的管道命令传值，下述awk脚本从du命令获得输入，并输出块和字节数。

```
$ pg duawk.awk
#!/bin/awk -f
# to call: du | duawk.awk
# prints file/direc's in bytes and blocks
BEGIN{
OFS="\t" ;
print "name" "\t\t","bytes","blocks\n"
print "==============================="}
{print $2,"\t\t",$1*512,$1}
```

为运行这段脚本，使用du命令，并管道输出至awk脚本。

```
$ du | awkdu.awk
name                    bytes    blocks
=======================================
./profile.d             2048     4
./X11                   135680   265
./rc.d/init.d           27136    53
./rc.d/rc0.d            512      1
./rc.d/rc1.d            512      1
```

9.2.13 awk数组

前面讲述split函数时，提到怎样使用它将元素划分进一个数组。这里还有一个例子：

```
$ awk 'BEGIN {print split("123#456#678", myarray, "#")}'
3
```

在上面的例子中，split返回数组myarray下标数。实际上myarray数组为：

Myarray[1]="123"
Myarray[2]="456"
Myarray[3]="678"

数组使用前，不必定义，也不必指定数组元素个数。经常使用循环来访问数组。下面是一种循环类型的基本结构：

```
For (element in array ) print array[element]
```

对于记录“123#456#678”，先使用split函数划分它，再使用循环打印各数组元素。操作脚本如下：

```
$ pg arraytest.awk
#!/bin/awk -f
# name: arraytest.awk
# prints out an array
BEGIN{
record="123#456#789";
split(record, myarray,"#") }
  END { for (i in myarray) {print myarray[i]}}
```

要运行脚本，使用/dev/null作为输入文件。

```
$ arraytest.awk  /dev/null
123
456
789
```

数组和记录

China-pub.com
下载

上面的例子讲述怎样通过split函数使用数组。也可以预先定义数组，并使用它与域进行比较测试，下面的例子中将使用更多的数组。

下面是从空手道数据库卸载的一部分数据，包含了学生级别及是否是成人或未成年人的信息，有两个域，分隔符为（#），文件如下：

```
$ pg grade_student.txt
Yellow#Junior
Orange#Senior
Yellow#Junior
Purple#Junior
Brown-2#Junior
White#Senior
Orange#Senior
Red#Junior
Brown-2#Senior
Yellow#Senior
Red#Junior
Blue#Senior
Green#Senior
Purple#Junior
White#Junior
```

脚本功能是读文件并输出下列信息。

1) 俱乐部中Yellow、Orange和Red级别的人各是多少。

2) 俱乐部中有多少成年人和未成年人。

查看文件，也许20秒内就会猜出答案，但是如果记录超过 60个又怎么办呢？这不会很容易就看出来，必须使用awk脚本。

首先看看awk脚本，然后做进一步讲解。

```
$ pg belts.awk
!/bin/awk -f
# name: belts.awk
# to call: belts.awk grade2.txt
# loops through the grade2.txt file and counts how many
# belts we have in (yellow, orange, red)
# also count how many adults and juniors we have
#
# start of BEGIN
# set FS and load the arrays with our values
BEGIN{FS="#"
# load the belt colours we are interested in only
belt["Yellow"]
belt["Orange"]
belt["Red"]
# end of BEGIN
# load the student type
student["Junior"]
student["Senior"]
}
# loop thru array that holds the belt colours against field-1
# if we have a match, keep a running total
  {for (colour in belt)
    {if ($1==colour)
    belt[colour]++}}
# loop thru array that holds the student type against
```

China-pub.com 下载

```
# field-2 if we have a match, keep a running total
  {for (senior_or_junior in student)
    {if ($2==senior_or_junior)
    student[senior_or_junior]++}}

# finished processing so print out the matches..for each array
END{ for (colour in belt) print "The club has", belt[colour], colour,
"Belts"

for (senior_or_junior in student) print "The club
has",student[senior_or_junior]\
,senior_or_junior, "students"}
```

BEGIN部分设置FS为符号#，即域分隔符，因为要查找Yellow、Orange和Red三个级别。然后在脚本中手工建立数组下标对学生做同样的操作。注意，脚本到此只有下标或元素，并没有给数组名本身加任何注释。初始化完成后，BEGIN部分结束。记住BEGIN部分并没有文件处理操作。

现在可以处理文件了。首先给数组命名为color，使用循环语句测试域1级别列是否等于数组元素之一（Yellow、Orange或Red），如果匹配，依照匹配元素将运行总数保存进数组。

同样处理数组‘Senior_or_junior’，浏览域2时匹配操作满足，运行总数存入junior或senior的匹配数组元素。

END部分打印浏览结果，对每一个数组使用循环语句并打印它。

注意在打印语句末尾有一个\符号，用来通知awk（或相关脚本）命令持续到下一行，当输入一个很长的命令，并且想分行输入时可使用这种方法。运行脚本前记住要加入可执行权限。

```
$ belts.awk grade_student.txt
The club has 2 Red Belts
The club has 2 Orange Belts
The club has 3 Yellow Belts
The club has 7 Senior students
The club has 8 Junior students
```

9.3 小结

awk语言学起来可能有些复杂，但使用它来编写一行命令或小脚本并不太难。本章讲述了awk的最基本功能，相信大家已经掌握了awk的基本用法。awk是shell编程的一个重要工具。在shell命令或编程中，虽然可以使用awk强大的文本处理能力，但是并不要求你成为这方面的专家。

China-pub.com
下载

第10章　sed用法介绍

sed是一个非交互性文本流编辑器。它编辑文件或标准输入导出的文本拷贝。标准输入可能是来自键盘、文件重定向、字符串或变量，或者是一个管道的文本。 sed可以做些什么呢？别忘了，Vi也是一个文本编辑器。sed可以随意编辑小或大的文件，有许多sed命令用来编辑、删除，并允许做这项工作时不在现场。sed一次性处理所有改变，因而变得很有效，对用户来讲，最重要的是节省了时间。

本章内容有：

• 抽取域。
• 匹配正则表达式。
• 比较域。
• 增加、附加、替换。
• 基本的sed命令和一行脚本。

可以在命令行输入sed命令，也可以在一个文件中写入命令，然后调用 sed，这与awk基本相同。使用sed需要记住的一个重要事实是，无论命令是什么， sed并不与初始化文件打交道，它操作的只是一个拷贝，然后所有的改动如果没有重定向到一个文件，将输出到屏幕。

因为sed是一个非交互性编辑器，必须通过行号或正则表达式指定要改变的文本行。

本章介绍sed用法和功能。本章大多编写的是一行命令和小脚本。这样做可以慢慢加深对sed用法的了解，取得宝贵的经验，以便最终自己编出大的复杂 sed脚本。

和grep与awk一样， sed是一种重要的文本过滤工具，或者使用一行命令或者使用管道与grep与awk相结合。

10.1　sed怎样读取数据

sed从文件的一个文本行或从标准输入的几种格式中读取数据，将之拷贝到一个编辑缓冲区，然后读命令行或脚本的第一条命令，并使用这些命令查找模式或定位行号编辑它。重复此过程直到命令结束。

10.2　调用sed

调用sed有三种方式：在命令行键入命令；将sed命令插入脚本文件，然后调用sed；将sed命令插入脚本文件，并使sed脚本可执行。

使用sed命令行格式为：

sed [选项] **sed**命令　输入文件。

记住在命令行使用sed命令时，实际命令要加单引号。sed也允许加双引号。

使用sed脚本文件，格式为：

sed [选项]　**-f**　**sed**脚本文件　输入文件

要使用第一行具有sed命令解释器的sed脚本文件，其格式为：

China-pub.com 下载

sed脚本文件 [选项] 输入文件

不管是使用shell命令行方式或脚本文件方式，如果没有指定输入文件， sed从标准输入中接受输入，一般是键盘或重定向结果。

sed选项如下：

n 不打印；sed不写编辑行到标准输出，缺省为打印所有行（编辑和未编辑）。p命令可以用来打印编辑行。

c 下一命令是编辑命令。使用多项编辑时加入此选项。如果只用到一条 sed命令，此选项无用，但指定它也没有关系。

f 如果正在调用sed脚本文件，使用此选项。此选项通知 sed一个脚本文件支持所有的 sed命令，例如：sed -f myscript.sed input_file，这里myscript.sed即为支持sed命令的文件。

10.2.1 保存sed输出

由于不接触初始化文件，如果想要保存改动内容，简单地将所有输出重定向到一个文件即可。下面的例子重定向 sed命令的所有输出至文件‘ myoutfile ’，当对结果很满意时使用这种方法。

```
$ sed 'some-sed-commands' input-file > myoutfile
```

10.2.2 使用sed在文件中查询文本的方式

sed浏览输入文件时，缺省从第一行开始，有两种方式定位文本：

1) 使用行号，可以是一个简单数字，或是一个行号范围。

2) 使用正则表达式，怎样构建这些模式请参见第 7章。

表10-1给出使用sed定位文本的一些方式。

表10-1 使用sed在文件中定位文本的方式

x	x为一行号，如1
x,y	表示行号范围从x到y，如2，5表示从第2行到第5行
/pattern/	查询包含模式的行。例如/disk/或/[a-z]/
/pattern/pattern/	查询包含两个模式的行。例如/disk/disks/
pattern/,x	在给定行号上查询包含模式的行。如/ribbon/,3
x,/pattern/	通过行号和模式查询匹配行。 3./vdu/
x,y!	查询不包含指定行号x和y的行。1,2!

10.2.3 基本sed编辑命令

表10-2列出了Sed的编辑命令。

表10-2 sed编辑命令

p	打印匹配行
=	显示文件行号
a\	在定位行号后附加新文本信息
i\	在定位行号后插入新文本信息
d	删除定位行
c\	用新文本替换定位文本

China-pub.com
下载

(续)

s	使用替换模式替换相应模式
r	从另一个文件中读文本
w	写文本到一个文件
q	第一个模式匹配完成后推出或立即推出
l	显示与八进制ASCII代码等价的控制字符
{}	在定位行执行的命令组
n	从另一个文件中读文本下一行，并附加在下一行
g	将模式2粘贴到/pattern n/
y	传送字符
n	延续到下一输入行；允许跨行的模式匹配语句

如果不特别声明，sed例子中使用下述文本文件quote.txt。

```
$ pg quote.txt
The honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
The local nurse Miss P.Neave was in attendance.
```

10.3 sed和正则表达式

sed识别任何基本正则表达式和模式及其行匹配规则。记住规则之一是：如果要定位一特殊字符，必须使用（\）屏蔽其特殊含义，如有必要请参照第 7章正则表达式。第7章使用的所有正则表达式在sed中都是合法的。

10.4 基本sed编程举例

下面通过例子实际检验一下sed的编辑功能。

10.4.1 使用p（rint）显示行

print命令格式为[address[，address]P。显示文本行必须提供sed命令行号。

```
$ sed '2p' quote.txt
The honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
The local nurse Miss P.Neave was in attendance.
```

错误在哪儿？原意只打印第二行，但是却打印了文件中所有行，为此需使用 -n选项，显示打印定位（匹配）行。

```
$ sed -n '2p' quote.txt
It was an evening of splendid music and company.
```

10.4.2 打印范围

可以指定行的范围，现打印1到3行，用逗号分隔行号。

```
$ sed -n '1,3p' quote.txt
The honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
```

China-pub.com 下载

10.4.3 打印模式

假定要匹配单词Neave，并打印此行，方法如下。使用模式 /pattern/格式，这里为/Neave/。

```
$ sed -n '/Neave/'p quote.txt
The local nurse Miss P.Neave was in attendance.
```

10.4.4 使用模式和行号进行查询

为编辑某个单词浏览一个文件时， sed返回包含指定单词的许多行。怎样使返回结果更精确以满足模式匹配呢？可以将行号和模式结合使用。下面这个例子，假定要改动文件 quote.txt最后一行中的单词the，使用sed查询the，返回两行：

```
$ sed -n '/The/'p quote.txt
The honeysuckle band played all night long for only $90.
The local nurse Miss P.Neave was in attendance.
```

使用模式与行号的混合方式可以剔除第一行，格式为 line_number,/pattern/。逗号用来分隔行号与模式开始部分。为达到预期结果，使用 4,/the/。意即只在第四行查询模式 the，命令如下：

```
$ sed -n '4,/The/'p quote.txt
The local nurse Miss P.Neave was in attendance.
```

10.4.5 匹配元字符

匹配元字符$前，必须使用反斜线\屏蔽其特殊含义。模式为/\$/ p。

```
$ sed -n '/\$/'p quote.txt
The honeysuckle band played all night long for only $90.
```

10.4.6 显示整个文件

要打印整个文件，只需将行范围设为第一行到最后一行 1,$。$意为最后一行。

```
$ sed -n '1,$p' quote.txt
The honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
The local nurse Miss P.Neave was in attendance.
```

10.4.7 任意字符

匹配任意字母，后跟任意字母的 0次或多次重复，并以ing结尾，模式为/.*ing/。可以使用这个模式查询以ing结尾的任意单词。

```
$ sed -n '/.*ing/'p quote.txt
It was an evening of splendid music and company.
```

10.4.8 首行

要打印文件第一行，使用行号：

```
$ sed -n '1p' quote.txt
The honeysuckle band played all night long for only $90.
```

10.4.9 最后一行

要打印最后一行，使用$。$是代表最后一行的元字符。

```
$ sed -n '$p' quote.txt
The local nurse Miss P.Neave was in attendance.
```

10.4.10 打印行号

要打印行号，使用等号=。打印模式匹配的行号，使用格式/pattern/=。

```
$ sed  -e '/music/=' quote.txt
The honeysuckle band played all night long for only $90.
2
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
The local nurse Miss P.Neave was in attendance.
```

整个文件都打印出来，并且匹配行打印了行号。如果只关心实际行号，使用 -e选项。

```
$ sed  -n '/music/=' quote.txt
2
```

如果只打印行号及匹配行，必须使用两个 sed命令，并使用e选项。第一个命令打印模式匹配行，第二个使用=选项打印行号，格式为sed -n -e /pattern/p -e /pattern/=。

```
$ sed -n -e '/music/p' -e '/music/=' quote.txt
It was an evening of splendid music and company.
2
```

10.4.11 附加文本

要附加文本，使用符号a\，可以将指定文本一行或多行附加到指定行。如果不指定文本放置位置，sed缺省放在每一行后面。附加文本时不能指定范围，只允许一个地址模式。文本附加操作时，结果输出在标准输出上。注意它不能被编辑，因为sed执行时，首先将文件的一行文本拷贝至缓冲区，在这里sed编辑命令执行所有操作（不是在初始文件上），因为文本直接输出到标准输出，sed并无拷贝。

要想在附加操作后编辑文本，必须保存文件，然后运行另一个 sed命令编辑它。这时文件的内容又被移至缓冲区。

附加操作格式如下：

```
[address]a\
text\
text\
...
text
```

地址指定一个模式或行号，定位新文本附加位置。 a\ 通知sed对a\后的文本进行实际附加操作。观察格式，注意每一行后面有一斜划线，这个斜划线代表换行。 sed执行到这儿，将创建一新行，然后插入下一文本行。最后一行不加斜划线， sed假定这是附加命令结尾。

当附加或插入文本或键入几个 sed命令时，可以利用辅助的 shell提示符以输入多行命令。这里没有这样做，因为可以留给使用者自己编写，并且在一个脚本文件中写这样的语句更适宜。现在马上讲述sed脚本文件。另外，脚本可以加入空行和注释行以增加可读性。

China-pub.com 下载

10.4.12 创建sed脚本文件

要创建脚本文件append.sed，输入下列命令：

```
$ pg append.sed
#!/bin/sed -f
/company/ a\
Then suddenly it happened.
```

保存它，增加可执行权限：

```
$ chmod u+x append.sed
```

运行，

```
$ append.sed quote.txt
The honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
Then suddenly it happened.
Too bad the disco floor fell through at 23:10.
The local nurse Miss P.Neave was in attendance.
```

如果返回‘ file not found ’，试在脚本前加入.\。

现在查看其具体功能。第一行是sed命令解释行。脚本在这一行查找sed以运行命令，这里定位在/bin。

第二行以/company/开始，这是附加操作起始位置。 a\通知sed这是一个附加操作，首先应插入一个新行。第三行是附加操作要加入到拷贝的实际文本。

输出显示附加结果。如果要保存输出，重定向到一个文件。

10.4.13 插入文本

插入命令类似于附加命令，只是在指定行前面插入。和附加命令一样，它也只接受一个地址。下面是插入命令的一般格式。地址是匹配模式或行号：

下面例子在以attendance结尾的行前插入文本utter confusion followed。

运行结果是：

也可以使用行号指定文本插入位置，插入位置在模式或指定行号 4之前。脚本如下：

```
#!/bin/sed -f
4 i\
Utter confusion followed.
```

10.4.14 修改文本

修改命令将在匹配模式空间的指定行用新文本加以替代，格式如下：

```
[address[,address] c\
text\
text\
text\
...
text
```

将第一行The honeysuckle band played all night long for only $90替换为The office Dibble band played well。首先要匹配第一行的任何部分，可使用模式‘ /Honeysuckle/ ’。sed脚本文件为change.sed。内容如下：

```
$ pg change.sed
#!/bin/sed -f
# change.sed
/honeysuckle/ c\
The Office Dibble band played well.
```

运行它，不要忘了给脚本增加可执行权限。 chmod u+x change.sed。

```
$ change.sed quote.txt
The Office Dibble band played well.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
The local nurse Miss P.Neave was in attendance.
```

像插入动作一样，可以使用行号代替模式，两种方式完成相同的功能。

```
#!/bin/sed -f
3 c\
The Office Dibble band played well.
```

可以对同一个脚本中的相同文件进行修改、附加、插入三种动作匹配和混合操作。

下面是一个带有注释的脚本例子。

```
$ pg mix.sed
#!/bin/sed -f
# this is a comment line, all comment starts with a #
# name: mix.sed


# this is the change on line 1
1 c\
The Dibble band were grooving.
# let's now insert a line
/evening/ i\
They played some great tunes.
# change the last line, a $ means last line
$ c\
Nurse Neave was too tipsy to help.
```

China-pub.com 下载

```
# stick in a new line before the last line
3 a\
Where was the nurse to help?
```

运行它，结果如下：

```
$ mix.sed quote.txt
The Dibble band were grooving.
They played some great tunes.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
Where was the nurse to help?
Nurse Neave was too tipsy to help.
```

10.4.15 删除文本

sed删除文本格式：

```
[address[, address]]d
```

地址可以是行的范围或模式，让我们看几个例子。

删除第一行；1d意为删除第一行。

```
$ sed '1d' quote.txt
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
The local nurse Miss P.Neave was in attendance.
```

删除第一到第三行：

```
$ sed '1,3d' quote.txt
The local nurse Miss P.Neave was in attendance.
```

删除最后一行：

```
$ sed '$d' quote.txt
The honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
```

也可以使用正则表达式进行删除操作。下面的例子删除包含文本‘ Neave’的行。

```
$ sed '/Neave/d' quote.txt
The honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
```

10.4.16 替换文本

替换命令用替换模式替换指定模式，格式为：

```
[address[, address]] s/ pattern-to-find /replacement-pattern/[g p w n]
```

s选项通知sed这是一个替换操作，并查询pattern-to-find，成功后用replacement-pattern替换它。

替换选项如下：

g 缺省情况下只替换第一次出现模式，使用g选项替换全局所有出现模式。

p 缺省sed将所有被替换行写入标准输出，加p选项将使-n选项无效。-n选项不打印输出结果。

w 文件名 使用此选项将输出定向到一个文件。

让我们看几个例子。替换 night为NIGHT，首先查询模式 night，然后用文本 NIGHT替换它。

```
$ sed 's/night/NIGHT/' quote.txt
The honeysuckle band played all NIGHT long for only $90.
```

要从 $90中删除 $符号（记住这是一个特殊符号，必须用 \屏蔽其特殊含义），在replacement-pattern部分不写任何东西，保留空白，但仍需要用斜线括起来。在 sed中也可以这样删除一个字符串。

```
$ sed 's/\$//' quote.txt
The honeysuckle band played all night long for only 90.
```

要进行全局替换，即替换所有出现模式，只需在命令后加 g选项。下面的例子将所有 The替换成Wow！。

```
$ sed 's/The/Wow!/g' quote.txt
Wow! honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
Too bad the disco floor fell through at 23:10.
Wow! local nurse Miss P.Neave was in attendance.
```

将替换结果写入一个文件用 w选项，下面的例子将 splendid替换为SPLENDID的替换结果写入文件sed.out：

```
$ sed s/splendid/SPLENDID/w sed.out' quote.txt
```

注意要将文件名括在sed的单引号里。文件结果如下：

```
$ pg sed.out
It was an evening of SPLENDID music and company.
```

10.5 使用替换修改字符串

如果要附加或修改一个字符串，可以使用（ &）命令，&命令保存发现模式以便重新调用它，然后把它放在替换字符串里面。这里给出一个修改的设计思路。先给出一个被替换模式，然后是一个准备附加在第一个模式后的另一个模式，并且后面带有 &，这样修改模式将放在匹配模式之前。例如，sed语句s/nurse/"Hello"&/p 的结果如下：

```
$ sed -n 's/nurse/"Hello" &/p' quote.txt
The local "Hello" nurse Miss P.Neave was in attendance.
```

原句是文本行The local nurse Miss P.Neave was in attendance。

记住模式中要使用空格，因为输出结果表明应加入空格。

还有一个例子：

```
$ sed -n 's/played/from Hockering &/p' quote.txt
The honeysuckle band from Hockering played all night long for only $90.
```

原句是The honeysuckle band played all night long for only $90。相信这种修改动作已经讲解得很清楚了。

10.6 将sed结果写入文件命令

像使用>文件重定向发送输出到一个文件一样，在 sed命令中也可以将结果输入文件。格式有点像使用替换命令：

China-pub.com 下载

```
[address[, address]]w filename
```

‘w’选项通知sed将结果写入文件。filename是自解释文件名。下面有两个例子。

文件quote.txt输出到屏幕。模式范围即1，2行输出到文件filedt。

下面例子中查询模式Neave，匹配结果行写入文件filedht。

10.7 从文件中读文本

处理文件时，sed允许从另一个文件中读文本，并将其文本附加在当前文件。此命令放在模式匹配行后，格式为：

```
address r filename
```

这里r通知sed将从另一个文件源中读文本。filename是其文件名。

现在创建一个小文件sedex.txt，内容如下：

```
$ pg sedex.txt
Boom boom went the music.
```

将sedex.txt内容附加到文件quote.txt的拷贝。在模式匹配行/company/后放置附加文本。本例为第三行。注意所读的文件名需要用单引号括起来。

```
$ sed '/company./r sedex.txt' quote.txt
The honeysuckle band played all night long for only $90.
It was an evening of splendid music and company.
Boom boom went the music.
Too bad the disco floor fell through at 23:10.
The local nurse Miss P.Neave was in attendance.
```

10.8 匹配后退出

有时需要在模式匹配首次出现后退出sed，以便执行其他处理脚本。退出命令格式为：

```
address q
```

下面的例子假定查询模式/.a.*/，意为任意字符后跟字符a，再跟任意字符0次或任意多次。查看文本文件，然后在下列行产生下列单词：

```
Line 1. band
Line 2. bad
Liner3. was
Line 4. was
```

查询首次出现模式，然后退出。需要将q放在sed语句末尾。

```
$ sed '/.a.*/q' quote.txt
The honeysuckle band played all night long for only $90.
```

China-pub.com
下载

10.9 显示文件中的控制字符

当从其他系统下载文件时，有时要删除整个文件的控制字符（非打印字符），从菜单中捕获一个应用的屏幕输出有时也会将控制字符输出进文件，怎样知道文件中是否有控制字符？使用cat -v filename命令，屏幕会乱叫，且到处都是一些垃圾字符，这可以确知文件中包含有控制字符，如果有兴趣可以观察一下这些字符以便于更加确认它们是控制字符。

一些系统中使用cat filename而不是cat-v来查看非打印字符。

sed格式为：

```
[address，[address]]l
```

‘l’意为列表。

一般情况下要列出整个文件，而不是模式匹配行，因此使用 l要从第一到最后一行。模式范围1，$即为此意。

如果cat一个文件，发现实际上包含有控制字符。

```
$ cat -v func.txt
This is is the F1 key:^[OP
This is the F2 key:^[OQ
```

现在运行sed命令，观察输出结果。

```
$ sed -n '1,$l' func.txt
This is is the F1 key:\033OP$
This is the F2 key:\033OQ$
```

sed找到并显示了两个控制字符。\033代表退格键，OP为F1键值，放在退格键后。第二行也是如此。

各系统控制字符键值可能不同，主要取决于其映射方式（例如使用 terminfo或termcap）。如果要在文本文件中插入控制字符 F1键，使用vi查看其键值，操作如下：

• 启动vi。
• 进入插入模式。
• 按下<Ctrl>键，然后按<v>键 （出现a^）。
• 释放上述两个键。
• 按下F1键（显示[OP]。
• 按下<ESC>键（显示F1键值）。

10.10 使用系统sed

前面已经讲述了sed的基本功能，但是在脚本或命令行中使用 sed真正要做的是修改或删除文件或字符串中文本。下面运用前面学过的知识讲述这一点。

10.10.1 处理控制字符

使用sed实现的一个重要功能是在另一个系统中下载的文件中剔除控制字符。

下面是传送过来的文件（dos.txt）的部分脚本。必须去除所有可疑字符，以便于帐号所有者使用文件。

```
$ pg dos.txt
```

China-pub.com 下载

```
12332##DISO##45.12^M
00332##LPSO##23.11^M
01299##USPD##34.46^M
```

可采取以下动作：

1) 用一个空格替换所有的（##）符号。

2) 删除起始域中最前面的0（00）。

3) 删除行尾控制字符（^M）。

一些系统中，回车符为^@和^L，如果遇到一些怪异的字符，不必担心，只要是在行尾并且全都相同就可以。

按步执行每一项任务，以保证在进行到下一任务前得到理想结果。使用输入文件 dos.txt。

任务1。 删除所有的#字符很容易，可以使用全局替换命令。这里用一个空格替换两个或更多的#符号。

```
$ sed 's/##*//g' dos.txt
12332 DISO 45.12^M
00332 LPSO 23.11^M
01299 USPD 34.46^M
```

任务2。 删除所有行首的0。使用^符号表示模式从行首开始， ^0*表示行首任意个0。模式s/^0*//g设置替换部分为空，即为删除模式，正是要求所在。

```
$ sed 's/^0*//g' dos.txt
12332##DISO##45.12^M
332##LPSO##23.11^M
1299##USPD##34.46^M
```

任务3。 最后去除行尾^M符号，为此需做全局替换。设置替换部分为空。模式为：‘s/^m//g’，注意‘^M’，这是一个控制字符。

要产生控制字符（^M），需遵从前面产生F1键同样的处理过程。步骤如下；键入 sed s/，然后按住<Ctrl>键和v键，释放v键，再按住^键，并保持<Ctrl>键不动，再释放两个键，最后按<return>键。下面命令去除行尾^M字符。

```
$ sed 's/^M//g' dos.txt
```

分步测试预想功能对理解整个过程很有帮助。用 sed在移到下一步前测试本步功能及结果很重要。如果不这样，可能会有一大堆包含怪异字符的意料外的结果。

将所有命令结合在一起，使用管道将 cat命令结果传入一系列sed命令，sed命令与上面几步精确过滤字符的sed相同。

```
$ cat dos.txt | sed 's/^0*//g' | sed 's/^M//g' | sed 's/##*/ /g'
12332 DISO 45.12
332 LPSO 23.11
1299 USPD 34.46
```

现在文件对帐号管理者可用。

可以将命令放在文件里，然后运行它。下面即为转换脚本。

```
$ pg dos.sed
#!/bin/sed -f
# name: dos.sed
# to call: dos.sed dos.txt

# get rid of the hash marks
```

China-pub.com
下载


```
s/##//g

# now get rid of the leading zeros
s/^0*//g
# now get rid of the carriage return
# the ^M is generated like we did for the F1 key.
s/\^M//g
```

通过仅指定一个sed命令可以将命令行缩短，本书后面部分介绍脚本中 sed的用法。

10.10.2 处理报文输出

当从数据库中执行语句输出时，一旦有了输出结果，脚本即可做进一步处理。通常先做一些整理，下面是一个sql查询结果。

```
Database      Size (MB)   Date Created
---------------------------------------------
GOSOUTH       2244        12/11/97
TRISUD        5632        8/9/99

(2 rows affected)
```

为了使用上述输出信息做进一步自动处理，需要知道所存数据库名称，为此需执行以下操作：

1) 使用s/-*//g删除横线------。
2) 使用/^$/d删除空行。
3) 使用$d删除最后一行
4) 使用1d删除第一行。
5) 使用awk {print $1}打印第一列。

命令如下，这里使用了cat，并管道传送结果到sed命令。

```
$ cat sql.txt | sed 's/--*//g' | sed '/^$/d' | sed '$d' | sed '1d' | awk
'{print $1}'
GOSOUTH
TRISUD
```

10.10.3 去除行首数字

对接下来卸载的这个文件实施的操作是去除行首所有数字，每个记录应以 UNH或UND开头，而不是UNH或UND前面的数字。文件如下：

```
$ pg UNH.txt
12345UND SPLLFC 234344
9999999UND SKKLT   3423
1UND SPLLY  434
...
```

使用基本正则表达式完成这个操作。 [0-9]代表行首任意数字，替换部分为空格是为了确保删除前面的匹配模式，即数字。

```
$ sed 's/^[0-9]//g' UNH.txt
UND SPLLFC 234344
UND SKKLT   3423
UND SPLLY  434
```

China-pub.com 下载

10.10.4 附加文本

当帐户完成设置一个文件时，帐号管理者可能要在文件中每个帐号后面加一段文字，下面是此类文件的一部分：

```
$ pg ok.txt
AC456
AC492169
AC9967
AC88345
```

任务是在每一行末尾加一个字符串‘ passed ’。

使用$命令修改各域会使工作相对容易些。首先需要匹配至少两个或更多的数字重复出现，这样将所有的帐号加进匹配模式。

```
$ sed 's/[0-9][0-9]*/& Passed/g' ok.txt
AC456 Passed
AC492169 Passed
AC9967 Passed
AC88345 Passed
```

10.10.5 从shell向sed传值

要从命令行中向sed传值，值得注意的是用双引号，否则功能不执行。

```
$ NAME="It's a go situation"
$ REPLACE="GO"
$ echo $NAME | sed "s/go/$REPLACE/g"
It's a GO situation
```

10.10.6 从sed输出中设置shell变量

从sed输出中设置 shell变量是一个简单的替换过程。运用上面的例子，创建 shell变量NEW-NAME，保存上述sed例子的输出结果。

```
$ NAME="It's a go situation"
$ REPLACE="GO"
$ NEW_NAME=`echo $NAME | sed "s/go/$REPLACE/g"`
$ echo $NEW_NAME
It's a GO situation
```

10.11 快速一行命令

下面是一些一行命令集。([]表示空格，[ ]表示tab键)

‘ s/\.$//g ’	删除以句点结尾行
‘ -e /abcd/d ’	删除包含abcd的行
‘ s/[][][]*/[]/g ’	删除一个以上空格，用一个空格代替
‘ s/^[][]*//g ’	删除行首空格
‘ s/\.[][]*/[]/g ’	删除句点后跟两个或更多空格，代之以一个空格
‘ /^$/d ’	删除空行
‘ s/^.//g ’	删除第一个字符
‘ s/COL\(...\)//g ’	删除紧跟COL的后三个字母
‘ s/^\///g ’	从路径中删除第一个\
‘ s/[]/[]//g ’	删除所有空格并用tab键替代
‘ S/^[]//g ’	删除行首所有tab键
‘ s/[]*//g ’	删除所有tab键

在结束这一章前，看看一行脚本的一些例子。

1. 删除路径名第一个\符号

将当前工作目录返回给sed，删除第一个\：

```
cd /usr/local
$ echo $PWD| sed 's/^\///g'
$ usr/local
```

2. 追加/插入文本

将"Mr Willis"字串返回给sed并在Mr后而追加"Bruce"。

```
$ echo "Mr Willis" | sed 's/Mr /& Bruce/g'
Mr Bruce Willis
```

3. 删除首字符

sed删除字符串“ accounts.doc ”首字符。

```
$ echo "accounts.doc" | sed 's/^.//g'
$ ccounts.doc
```

4. 删除文件扩展名

sed删除“ accounts.doc ”文件扩展名。

```
$ echo "accounts.doc" | sed 's/.doc//g'
accounts
```

5. 增加文件扩展名

sed附加字符串“ .doc ”到字符串“ accounts ”。

```
$ echo "accounts" | sed 's/$/.doc/g'
accounts.doc
```

6. 替换字符系列

如果变量x含有下列字符串：

```
$ x="Department+payroll%Building G"
$ echo $x
$ Department+payroll%Building G
```

如果要实现下列转换：

```
+ to 'of'
% to 'located'
```

sed命令是：

```
$ echo $x | sed 's/\+/ of /g' | sed 's/\%/ Located at /g'
Department of payroll Located Building G
```

10.12 小结

sed是一个强大的文本过滤工具。使用sed可以从文件或字符串中抽取所需信息。正像前面讲到的，sed不必写太长的脚本以取得所需信息。本章只讲述了sed的基本功能，但使用这些功能就可以执行许多任务了。

如果使用sed对文件进行过滤，最好将问题分成几步，分步执行，且边执行边测试结果。经验告诉我们，这是执行一个复杂任务的最有效方式。

China-pub.com
下载

第11章 合并与分割

几年前，我习惯于使用运行在终端的 PICK操作的UNIX集合，我实际使用 PICK应用的大部分时间花费在分类与连接过程中，且使用极其频繁。很幸运我没有成为一个全职的 PICK操作员。

有几种工具用来处理文本文件分类、合并和分割操作，本章详细介绍这些工具。

本章内容有：

• 实用的分类（sort）操作。
• uniq。
• join。
• cut 。
• paste。
• split。

11.1 sort用法

sort命令将许多不同的域按不同的列顺序分类。当查阅注册文件或为另一用户对下载文件重排文本列时， sort工具很方便。实际上，使用其他 UNIX工具时，已假定工作文件已经被分过类。无论如何，分类文件比不分类文件看起来更有意义。

11.1.1 概述

UNIX/LINUX自带的sort功能很强大。尽管有时在使用 sort各种不同的选项时人们已经很小心，但仍会产生意想不到的结果。 sort选项很长，甚至有时在各种不同开关的实际功能和结果进行比较时也会遇到麻烦，原因可能是在结合使用 sort的不同选项时有些概念模糊不清。

本章不讨论各种不同的 sort方法（不能说sort不够强大；它很慢，但观察数值交替变化是很有趣的）也不讨论各种不同开关的结合使用功效。本章只讲到主要的 sort选项，伴随有大量实例。与sort结合使用的uniq、join、cut和paste方法与split方法也将会涉及到。

上面提到， sort命令选项很长，下面介绍本章使用的各种选项。

11.1.2 sort选项

sort命令的一般格式为：

```
sort -cmu -o output_file [other options] +pos1 +pos2 input_files
```

下面简要介绍一下sort的参数：

-c 测试文件是否已经分类。

-m 合并两个分类文件。

-u 删除所有复制行。

-o 存储sort结果的输出文件名。

其他选项有：

-b 使用域进行分类时，忽略第一个空格。

-n 指定分类是域上的数字分类。

-t 域分隔符；用非空格或tab键分隔域。

-r 对分类次序或比较求逆。

+n n为域号。使用此域号开始分类。

n n为域号。在分类比较时忽略此域，一般与 +n一起使用。

post1 传递到m，n。m为域号，n为开始分类字符数；例如4，6意即以第5域分类，从第7个字符开始。

11.1.3 保存输出

-o选项保存分类结果，然而也可以使用重定向方法保存。下面例子保存结果到 results.out：

```
$ sort video.txt >results.out
```

11.1.4 sort启动方式

缺省情况下，sort认为一个空格或一系列空格为分隔符。要加入其他方式分隔，使用 -t选项。

sort执行时，先查看是否为域分隔设置了 -t选项，如果设置了，则使用它来将记录分隔成域0、域1等等；如果未设置，用空格代替。缺省时 sort将整个行排序，指定域号的情况例外。

下面是文件video.txt的清单，包含了上个季度家电商场的租金情况。各域为：(1）名称，(2）供货区代码，(3）本季度租金，(4）本年租金。域分隔符为冒号。为此对此例需使用‘ -t ’选项。文件如下：

```
$ pg video.txt
Boys in Company C:HK:192:2192
Alien:HK:119:1982
The Hill:KL:63:2972
Aliens:HK:532:4892
Star Wars:HK:301:4102
A Few Good Men:KL:445:5851
Toy Story:HK:239:3972
```

11.1.5 sort对域的参照方式

关于sort的一个重要事实是它参照第一个域作为域 0，域1是第二个域，等等。 sort也可以使用整行作为分类依据。为防止混淆，对于此文件用户应按如下方式参照域并做分类依据：

Field 0	Field 1	Field 2	Field 3
Star Wars	HK	301	4102
A Few Good Men	KL	445	5851

sort将定位各域，因此应把域0作为分类键0，域1作为分类键1等等。

11.1.6 文件是否已分类

怎样分辨文件是否已分类？如果只有30行，看看就知道了，但如果是400行呢，使用sort-c

China-pub.com 下载

通知sort文件是否按某种顺序分类。

```
$ sort -c video.txt
sort: disorder on video.txt
```

结果显示未分类，现在将之分类，再试一次：

```
$ sort -c video.txt
$
```

返回提示符表明已分类。然而如果测试成功，返回一个信息行会更好。

11.1.7 基本sort

最基本的sort方式为sort filename，按第一域进行分类（分类键0）。实际上读文件时sort操作将行中各域进行比较，这里返回基于第一域 sort的结果，如下所示：

```
$ sort -t: video.txt
Alien:HK:119:1982
Aliens:HK:532:4892
Boys in Company C:HK:192:2192
A Few Good Men:KL:445:5851
Star Wars:HK:301:4102
The Hill:KL:63:2972
Toy Story:HK:239:3972
```

11.1.8 sort分类求逆

如果要逆向sort结果，使用-r选项。在通读大的注册文件时，使用逆向 sort很方便。下面是按域0分类的逆向结果。

```
$ sort -t: -r video.txt
Toy Story:HK:239:3972
The Hill:KL:63:2972
Star Wars:HK:301:4102
A Few Good Men:KL:445:5851
Boys in Company C:HK:192:2192
Aliens:HK:532:4892
Alien:HK:119:1982
```

11.1.9 按指定域分类

有时需要只按第 2域（分类键 1）分类。这里为重排报文中供应区代码，使用 t1，意义为按分类键1分类。下面的例子中，所有供应区代码按分类键 1分类；注意分类键 2和3对应各域也被分类。

```
$ sort -t: +1  video.txt
Alien:HK:119:1982
Boys in Company C:HK:192:2192
Toy Story:HK:239:3972
Star Wars:HK:301:4102
Aliens:HK:532:4892
A Few Good Men:KL:445:5851
The Hill:KL:63:2972
```

11.1.10 数值域分类

依此类推，要按第三分类键分类，使用 t3。但是因为这是数值域，即为数值分类，可以使

用-n选项。下面例子为按年租金分类命令及结果：

```
$ sort -t: +3n  video.txt
Alien:HK:119:1982
Boys in Company C:HK:192:2192
The Hill:KL:63:2972
Toy Story:HK:239:3972
Star Wars:HK:301:4102
Aliens:HK:532:4892
A Few Good Men:KL:445:5851
```

如果不加-n，结果会怎样？这里假定按第3域分类，找出最好的季度租金。因为是分类键2，所以使用t2。

```
$ sort -t: +2 video.txt
Alien:HK:119:1982
Boys in Company C:HK:192:2192
Toy Story:HK:239:3972
Star Wars:HK:301:4102
A Few Good Men:KL:445:5851
Aliens:HK:532:4892
The Hill:KL:63:2972
```

观察结果，分类进行了，但不是预想的结果，因为第3域为数值域。当然这个结果也是某种类型的排列，录像机The Hill应该在第二行，但结果是：sort只查看第3域每个数值的第一个数，并按其分类，然后再按第二个数依次下去。

记住按数值域分类要加-n，这样才会得到预想结果。

```
$ sort -t: +2n video.txt
The Hill:KL:63:2972
Alien:HK:119:1982
Boys in Company C:HK:192:2192
Toy Story:HK:239:3972
Star Wars:HK:301:4102
A Few Good Men:KL:445:5851
Aliens:HK:532:4892
```

现在对了，可以看出本季度卖点最高的是Aliens。如果使用-r选项，将会把Aliens放在第一行。

11.1.11 唯一性分类

有时，原文件中有重复行，这时可以使用-u选项进行唯一性（不重复）分类以去除重复行，本例中Alien有相同的两行。带重复行的文件如下，其中Alien插入了两次：

```
$ pg video.txt
Boys in Company C:HK:192:2192
Alien:HK:119:1982
The Hill:KL:63:2972
Aliens:HK:532:4892
Star Wars:HK:301:4102
A Few Good Men:KL:445:5851
Alien:HK:119:1982
```

使用-u选项去除重复行，不必加其他选项，sort会自动处理。

```
$ sort -u video.txt
Alien:HK:119:1982
Aliens:HK:532:4892
Boys in Company C:HK:192:2192
```

China-pub.com 下载

```
A Few Good Men:KL:445:5851
Star Wars:HK:301:4102
The Hill:KL:63:2972
```

11.1.12 使用k的其他sort方法

sort还有另外一些方法指定分类键。可以指定 k选项，第1域（分类键）以1开始。不要与前面相混淆。我经常使用这个选项。因为我习惯于第一域为数值 1，这样使用sort时用同样的数值做分类依据会更有意义。其他选项也可以使用 k，主要用于指定分类域开始的字符数目。

要在第1域进行分类，可以使用-k4，这是按年租金分类的次序。

```
$ sort -t: -k4  video.txt
Alien:HK:119:1982
Boys in Company C:HK:192:2192
The Hill:KL:63:2972
Star Wars:HK:301:4102
Aliens:HK:532:4892
A Few Good Men:KL:445:5851
```

11.1.13 使用k做分类键排序

可以指定分类键次序。先以第 4域，再以第1域分类，命令为-k4 -k1，也可以反过来，以便在文件首行显示最高年租金，方法如下：

```
$ sort -t: -r -k4 -k1 video.txt
A Few Good Men:KL:445:5851
Aliens:HK:532:4892
Star Wars:HK:301:4102
The Hill:KL:63:2972
Boys in Company C:HK:192:2192
Alien:HK:119:1982
```

11.1.14 指定sort序列

可以指定分类键顺序，也可以使用 -n选项指定不使用哪个分类键进行查询。看下面的 sort 命令：

sort +0 -2 +3

该命令意即开始以域0分类，忽略域2，然后再使用域3分类。

11.1.15 pos用法

指定开始分类的域位置的另一种方法是使用如下格式：

sort +field_number .characters_in

意即从field_number开始分类，但是要在此域的第 characters_in个字符开始。

这里是一个例子，供应区代码加入一些后缀。如：

```
$ pg video.txt
Boys in Company C:HK48:192:2192
Alien:HK57:119:1982
The Hill:KL23:63:2972
Aliens:HK11:532:4892
```

```
Star Wars:HK38:301:4102
A Few Good Men:KL87:445:5851
Toy Story:HK65:239:3972
```

要只使用供应区代码后缀部分将文件分类，其命令为 +1.2，意即以第1域最左边第3个字符开始分类，其具体含义及脚本如下：

	Field 0	Field 1	Field 2	Field 3
	Aliens	H K 1 1	532	4892
Characters in:		0 1 2 3		

```
$ sort -t: +1.2 video.txt
Aliens:HK11:532:4892
The Hill:KL23:63:2972
Star Wars:HK38:301:4102
Boys in Company C:HK48:192:2192
Alien:HK57:119:1982
Toy Story:HK65:239:3972
A Few Good Men:KL87:445:5851
```

11.1.16 使用head和tail将输出分类

分类操作时，不一定要显示整个文件或一页以查看 sort结果中的第一和最后一行。如果只显示最高年租金，按第 4域分类-k4并求逆，然后使用管道只显示 sort输出的第一行，此命令为head，可以指定查阅行数。如果只有第一行，则为 head -1：

```
$ sort -t: -r -k4  video.txt | head -1
A Few Good Men:KL:445:5851
```

要查阅最低年租金，使用tail命令与head命令刚好相反，它显示文件倒数几行。 1为倒数一行，2为倒数两行等等。查阅最后一行为 tail -1。结合上述的sort命令和tail命令显示最低年租金：

```
$ sort -t: -r -k4 video.txt | tail -1
Alien:HK:119:1982
```

可以使用head或tail查阅任何大的文本文件， head用来查阅文件头，基本格式如下：

head [how_many_lines_to_display] file_name

Tail用来查阅文件尾，基本格式为：

tail [how_many_lines_to_display] file_name

如果使用head或tail时想省略显示行数，缺省时显示 10行。

要查阅文件前20行：

```
$ head -20 file_name
```

要查阅文件后7行：

```
$ tail -7 file_name
```

11.1.17 awk使用sort输出结果

对数据分类时，对 sort结果加一点附加信息很有必要，对其他用户尤其如此。使用 awk可以轻松完成这一功能。比如说采用上面最低租金的例子，需要将 sort结果管道输出到awk，不要忘了用冒号作域分隔符，显示提示信息和实际数据。

China-pub.com 下载

```
$ sort -t: -r -k4 video.txt|tail -1 | awk -F: '{print "Worst rental
",$1,"has been rented "$3}'

Worst rental Alien has been rented 119
```

11.1.18 将两个分类文件合并

将文件合并前，它们必须已被分类。合并文件可用于事务处理和任何种类的修改操作。下面这个例子，因为忘了把两个家电名称加入文件，它们被放在一个单独的文件里，现在将之并入一个文件。分类的合并格式为‘ sort -m sorted_file1 sorted_file2，下面是包含两个新家电名称的文件列表，它已经分类完毕：

```
$ pg video2.txt
Crimson Tide:134:2031
Die Hard:152:2981
```

使用-m +o。将这个文件并入已存在的分类文件 video.sort，要以名称域进行分类，实际上没有必要加入+o，但为了保险起见，还是加上的好。

```
$ sort -t: -m +0 video2.txt video.sort
Alien:HK:119:1982
Aliens:HK:532:4892
Boys in Company C:HK:192:2192
Crimson Tide:134:2031
Die Hard:152:2981
A Few Good Men:KL:445:5851
Star Wars:HK:301:4102
The Hill:KL:63:2972
```

11.2 系统sort

sort可以用来对/etc/passwd文件中用户名进行分类。这里需要以第 1域即注册用户名分类，然后管道输出结果到awk，awk打印第一域。

```
$ cat passwd | sort -t: +0 | awk -F":" '{print $1}'
adm
bin
daemon
...
...
```

sort还可以用于df命令，以递减顺序打印使用列。下面是一般 df输出。

```
$ df
Filesystem    1024-blocks  Used    Available Capacity Mounted on
/dev/hda5     495714       291027  179086    62%      /
/dev/hda1     614672       558896  55776     91%      /dos
```

使用-b选项，忽略分类域前面的空格。使用域 4（+4），即容量列将分类求逆，最后得出文件系统自由空间的清晰列表。

```
$ df | sort -b -r +4
Filesystem    1024-blocks  Used    Available Capacity Mounted on
/dev/hda1     614672       558896  55776     91%      /dos
/dev/hda5     495714       291027  179086    62%      /
```

在一个文本文件中存入所有 IP地址的拷贝，这样查看本机 IP地址更容易一些。有时如果在管理员权限下，就需要将此文件分类。将 IP地址按文件中某种数值次序分类时，需要指定

域分隔符为句点。这里只需关心IP地址的最后一段。分类应从此域即域3开始，未分类文件如下：

```
$ pg iplist
193.132.80.123  dave tansley
193.132.80.23   HP printer 2nd floor
193.132.80.198  JJ. Peter's scanner
193.132.80.38   SPARE
193.132.80.78   P.Edron
```

分类后结果如下：

```
$ sort -t.  +3n iplist
193.132.80.23   HP printer 2nd floor
193.132.80.38   SPARE
193.132.80.78   P.Edron
193.132.80.123  dave tansley
193.132.80.198  JJ. Peter's scanner
```

11.3 uniq用法

uniq用来从一个文本文件中去除或禁止重复行。一般uniq假定文件已分类，并且结果正确。我们并不强制要求这样做，如果愿意，可以使用任何非排序文本，甚至是无规律行。

可以认为uniq有点像sort命令中唯一性选项。对，在某种程度上讲正是如此，但两者有一个重要区别。sort的唯一性选项去除所有重复行，而uniq命令并不这样做。重复行是什么？在uniq里意即持续不断重复出现的行，中间不夹杂任何其他文本，现举例如下：

```
$ pg myfile.txt
May Day
May Day
May Day
Going Down
May Day.
```

uniq将前三个May Day看作重复副本，但是因为第4行有不同的文本，故并不认为第五行持续的May Day为其副本。uniq将保留这一行。

命令一般格式：

uniq -u d c -f input-file output-file

其选项含义：

-u 只显示不重复行。

-d 只显示有重复数据行，每种重复行只显示其中一行

-c 打印每一重复行出现次数。

-f n为数字，前n个域被忽略。

一些系统不识别-f选项，这时替代使用-n。

使用本节开始时的文本，创建文件myfile.txt，在此文件上运行uniq命令。

```
$ uniq myfile.txt
May Day
Going Down
May Day
```

注意第5行保留下来，其文本为最后一行May Day。如果运行sort -u，将只返回May Day和Going Down。

China-pub.com 下载

连续重复出现

使用-c选项显示行数，即每个重复行数目。本例中，行 May Day重复出现三次。

```
$ uniq -c myfile.txt
      3 May Day
      1 Going Down
      1 May Day
```

1. 不唯一

使用-d显示重复出现的不唯一行：

```
$ uniq -d myfile.txt
May Day
```

2. 对特定域进行测试

使用-n只测试一行一部分的唯一性。例如 -5意即测试第5域后各域唯一性。域从 1开始记数。

如果忽略第1域，只测试第2域唯一性，使用-n2，下述文件包含一组数据，其中第 2域代表组代码。

```
$ pg parts.txt
AK123 OP
DK122 OP
EK999 OP
```

运行uniq，将返回所有行。因为这个文件每一行都不同。

```
$ uniq -c parts.txt
      1 AK123 OP
      1 DK122 OP
      1 EK999 OP
```

如果指定测试在第1域后，结果就会不同。uniq会比较三个相同的OP，因此将返回一行。

```
$ uniq -f2 parts.txt
AK123 OP
```

如果‘ -f ’返回错误，替代使用：

```
$ uniq -n2 parts.txt
AK123 OP
```

11.4 join用法

join用来将来自两个分类文本文件的行连在一起。如果学过 SQL语言，可能会很熟悉join命令。

下面讲述join工作方式。这里有两个文件 file1和file2，当然已经分类。每个文件里都有一些元素与另一个文件相关。由于这种关系， join将两个文件连在一起，这有点像修改一个主文件，使之包含两个文件里的共同元素。

文本文件中的域通常由空格或 tab键分隔，但如果愿意，可以指定其他的域分隔符。一些系统要求使用join时文件域要少于20，为公平起见，如果域大于20，应使用DBMS系统。

为有效使用join，需分别将输入文件分类。

其一般格式为：

join [options] input-file1 input-file2.

China-pub.com
下载

让我们看看它的可用选项列表：

an n为一数字，用于连接时从文件 n中显示不匹配行。例如，-a1显示第一个文件的不匹配行，-a2为从第二个文件中显示不匹配行。

o n.m n为文件号，m为域号。1.3表示只显示文件1第三域，每个n，m必须用逗号分隔，如1.3，2.1。

j n m n为文件号，m为域号。使用其他域做连接域。

t 域分隔符。用来设置非空格或tab键的域分隔符。例如，指定冒号做域分隔符 -t：。

现有两个文本文件，其中一个包含名字和街道地址，称为 name.txt，另一个是名字和城镇，为town.txt。

```
$ pg names.txt
M.Golls 12 Hidd Rd
P.Heller The Acre
P.Willey 132 The Grove
T.Norms 84 Connaught Rd
K.Fletch 12 Woodlea

$ pg town.txt
M.Golls Norwich NRD
P.Willey Galashiels GDD
T.Norms Brandon BSL
K.Fletch Mildenhall MAF
```

连接两个文件

连接两个文件，使得名字支持详细地址。例如 M.Golls记录指出地址为12 Hidd Rd。连接域为域0——名字域。因为两个文件此域相同，join将假定这是连接域：

```
$ join names.txt town.txt
M.Golls  12 Hidd Rd Norwich NRD
P.Willey 132 The Grove Galashiels GDD
T.Norms  84 Connaught Rd Brandon BSL
K.Fletch 12 Woodlea Mildenhall MAF
```

好，工作完成。缺省join删除或去除连接键的第二次重复出现，这里即为名字域。

1. 不匹配连接

如果一个文件与另一个文件没有匹配域时怎么办？这时 join不可以没有参数选项，经常指定两个文件的-a选项。下面的例子显示匹配及不匹配域。

```
$ join -a1  -a2 names.txt town.txt
M.Golls  12 Hidd Rd Norwich NRD
P.Heller The Acre
P.Willey 132 The Grove Galashiels GDD
T.Norms  84 Connaught Rd Brandon BSL
K.Fletch 12 Woodlea Mildenhall MAF
```

输出表明P.Heller不匹配第二个文件中任何一个记录。再运行这个命令，但指定只显示第一个文件中不匹配行：

```
$ join -a1 names.txt town.txt
```

2. 选择性连接

使用-o选项选择连接域。例如要创建一个文件仅包含人名及城镇，join执行时需要指定显示域。方式如下：

使用1.1显示第一个文件第一个域，2.2显示第二个文件第二个域，其间用逗号分隔。命令为：

China-pub.com 下载

```
$ join  -o 1.1,2.2 names.txt town.txt
M.Golls Norwich
P.Willey Galashiels
T.Norms Brandon
K.Fletch Mildenhall
```

要创建此新文件，将输出结果重定向到一个文件即可。

```
$ join  -o 1.1,2.2 names.txt town.txt >towns.txt
```

使用-jn m进行其他域连接，例如用文件1域3和文件域2做连接键，命令为：

```
join -j1 3 -j2 2 file1 file2
```

下面观察一个具体实例。有两个文件：

```
$ pg pers
P.Jones Office Runner   ID897
S.Round UNIX admin      ID666
L.Clip  Personl Chief   ID982

$ pg pers2
Dept2C  ID897   6 years
Dept3S  ID666   2 years
Dept5Z  ID982   1 year
```

文件pers包括名字、工作性质和个人 ID号。文件pers2包括部门、个人ID号及工龄。连接应使用文件pers中域4，匹配文件pers2中域2，命令及结果如下：

```
$ join -j1 4 -j2 2 pers pers2
ID897 P.Jones Office Runner Dept2C 6 years
ID666 S.Round UNIX admin Dept3S    2 years
ID982 L.Clip Personl Chief Dept5Z  1 year
```

使用join应注意连接域到底是哪一个，比如说你认为正在访问域 4，但实际上join应该访问域5，这样将不返回任何结果。如果是这样，用 awk检查域号。例如，键入$ awk '{print $4}'文件名，观察其是否匹配假想域。

11.5 cut用法

cut用来从标准输入或文本文件中剪切列或域。剪切文本可以将之粘贴到一个文本文件。下一节将介绍粘贴用法。

cut一般格式为：

```
cut [options] file1 file2
```

下面介绍其可用选项：

-c list 指定剪切字符数。

-f field 指定剪切域数。

-d 指定与空格和tab键不同的域分隔符。

-c用来指定剪切范围，如下所示：

-c1，5-7 剪切第1个字符，然后是第5到第7个字符。

-c1-50 剪切前50个字符。

-f 格式与-c相同。

-f 1，5 剪切 第1域，第5域。

-f1，10-12 剪切第1域，第10域到第12域。

参照上一节中的文件‘ pers ’，现在从'pers'文件中剪切文本。使用冒号做其域分隔符。

China-pub.com
下载


```
$ pg pers
P.Jones:Office Runner:ID897
S.Round:UNIX admin:ID666
L.Clip:Personl Chief:ID982
```

11.5.1 使用域分隔符

文件中使用冒号“ :”为域分隔符，故可用-d选项指定冒号，如-d :。如果有意观察第3域，可以使用-f3。要抽取ID域。可使用命令如下：

```
$ cut -d: -f3 pers
ID897T
ID666
ID982
```

11.5.2 剪切指定域

cut命令中剪切各域需用逗号分隔，如剪切域 1和3，即名字和ID号，可以使用：

```
$ cut -d: -f1,3 pers
P.Jones:ID897
S.Round:ID666
L.Clip:ID982
```

要从文件/etc/passwd中剪切注册名及缺省根目录，需抽取域 1和域3：

```
$ cut -d: -f1,6 /etc/passwd
gopher:/usr/lib/gopher-data
ftp:/home/ftp
peter:/home/apps/peter
dave:/home/apps/dave
...
```

使用-c选项指定精确剪切数目。这种方法需确切知道开始及结束字符。通常我不用这种方法，除非在固定长度的域或文件名上。

当信息文件传送到本机时，查看部分文件名就可以识别文件来源。要得到这条信息需抽取文件名后三个字符。然后才决定将之存在哪个目录下。下面的例子显示文件名列表及相应cut命令：

```
2231DG
2232DP
2236DK

$ ls 223*|cut -c4-6
1DG
2DP
6DK
```

如果使用ls -l命令作部分输出，情况将不同。需使用 -c选项。

```
-rw-r--r--   1 dave   admin      56 Apr 26 20:40 tr2.txt
-rw-r--r--   1 dave   admin      71 Apr 26 21:20 trpro.txt
```

要剪切字符，须计算 ls -l列表中的字符数。如显示权限用 cut -c1-10。然而这种方法可能相当慢，因此需要使用其他工具将相应信息抽取出来。要剪切谁正在使用系统的用户信息，方法如下：

```
$ who -u| cut -c1-8
root
dave
.
```

China-pub.com 下载

```
peter
```

11.6 paste用法

cut用来从文本文件或标准输出中抽取数据列或者域，然后再用 paste可以将这些数据粘贴起来形成相关文件。粘贴两个不同来源的数据时，首先需将其分类，并确保两个文件行数相同。

paste将按行将不同文件行信息放在一行。缺省情况下， paste连接时，用空格或tab键分隔新行中不同文本，除非指定-d选项，它将成为域分隔符。

paste格式为；

```
paste -d -s -file1 file2
```

选项含义如下：

-d 指定不同于空格或tab键的域分隔符。例如用@分隔域，使用-d@。

-s 将每个文件合并成行而不是按行粘贴。

\- 使用标准输入。例如ls -l |paste ，意即只在一列上显示输出。

从前面的剪切中取得下述两个文件：

```
$ pg pas1
ID897
ID666
ID982

$ pg pas2
P.Jones
S.Round
L.Clip
```

基本paste命令将之粘贴成两列：

```
$ paste pas1 pas2
ID897    P.Jones
ID666    S.Round
ID982    L.Clip
```

11.6.1 指定列

通过交换文件名即可指定哪一列先粘：

```
$ paste pas2 pas1
P.Jones ID897
S.Round ID666
L.Clip  ID982
```

11.6.2 使用不同的域分隔符

要创建不同于空格或tab键的域分隔符，使用-d选项。下面的例子用冒号做域分隔符。

```
$ paste -d: pas2 pas1
P.Jones:ID897
S.Round:ID66
```

要合并两行，而不是按行粘贴，可以使用-s选项。下面的例子中，第一行粘贴为名字，第二行是ID号。

```
$ paste -s pas2 pas1
P.Jones S.Round L.Clip
ID897   ID666   ID982
```

11.6.3 paste命令管道输入

paste命令还有一个很有用的选项（-）。意即对每一个（-），从标准输入中读一次数据。使用空格作域分隔符，以一个4列格式显示目录列表。方法如下：

```
$ pwd
$ /etc
$ ls | paste -d" " - - - -
init.d rc rc.local rc.sysinit
rc0.d rc1.d rc2.d rc3.d
rc4.d rc5.d rc6.d
```

也可以以一列格式显示输出：

```
$ ls | paste -d"" -
init.d
rc
rc.local
rc.sysinit
rc0.d
rc1.d
...
```

11.7 split用法

split用来将大文件分割成小文件。有时文件越来越大，传送这些文件时，首先将其分割可能更容易。使用vi或其他工具诸如sort时，如果文件对于工作缓冲区太大，也会存在一些问题。因此有时没有选择余地，必须将文件分割成小的碎片。

split命令一般格式：

```
split -output_file-size input-filename output-filename
```

这里output-file-size指的是文本文件被分割的行数。split查看文件时，output-file-size选项指定将文件按每个最多1000行分割。如果有个文件有2800行，那么将分割成3个文件，分别有1000、1000、800行。每个文件格式为x[aa]到x[zz]，x为文件名首字母，[aa]、[zz]为文件名剩余部分顺序字符组合，下面的例子解释这一点。

假定文件bigone.txt有2800行，split命令产生下列文件：

```
$ split bigone.txt
xaa
xab
xac
```

文件大小为：

Size	Filename
1000	xaa
1000	xab
800	xac

可以使用output-file-size选项来分割文件。以下为一个6行文件。

```
$ pg split1
this is line1
```

China-pub.com 下载

```
this is line2
this is line3
this is line4
this is line5
this is line6
```

按每个文件2行分割，命令为：

```
$ split -2 split1
```

观察其结果。

```
$ ls -lt |head
total 205
-rw-r--r--   1 dave     admin          28 Apr 30 13:12 xaa
-rw-r--r--   1 dave     admin          28 Apr 30 13:12 xab
-rw-r--r--   1 dave     admin          28 Apr 30 13:12 xac
   ...
```

文件有6行，split按每个文件两行进行了分割，并按字母顺序命名文件。为进一步确信操作成功，观察一个新文件内容：

```
$ pg xac
this is line5
this is line6
```

11.8 小结

本章讲述了对文本文件进行基本的合并和分割处理的各种工具。诸如 sort、join、split、uniq、cut和paste，并附有大量实例。使用这些工具将使你事半功倍。现在如果遇到一个未处理文件，相信你已知道使用什么工具将数据转化为更有意义的信息。

China-pub.com
下载

第12章 tr 用 法

12.1 关于tr

tr用来从标准输入中通过替换或删除操作进行字符转换。 tr主要用于删除文件中控制字符或进行字符转换。使用tr时要转换两个字符串：字符串1用于查询，字符串2用于处理各种转换。tr刚执行时，字符串1中的字符被映射到字符串2中的字符，然后转换操作开始。

本章内容有：

- 大小写转换。
- 去除控制字符。
- 删除空行。

带有最常用选项的tr命令格式为：

```
tr-c-d-s["string1_to_translate_from"]["string2_to_triampsulta_te_to"]
file
```

这里：

-c 用字符串1中字符集的补集替换此字符集，要求字符集为 ASCII。

-d 删除字符串1中所有输入字符。

-s 删除所有重复出现字符序列，只保留第一个；即将重复出现字符串压缩为一个字符串。

Input-file是转换文件名。虽然可以使用其他格式输入，但这种格式最常用。

12.1.1 字符范围

使用tr时，可以指定字符串列表或范围作为形成字符串的模式。这看起来很像正则表达式，但实际上不是。指定字符串1或字符串2的内容时，只能使用单字符或字符串范围或列表。

[a-z] a-z内的字符组成的字符串。

[A-Z] A-Z内的字符组成的字符串。

[0-9] 数字串。

/octal 一个三位的八进制数，对应有效的ASCII字符。

[O*n] 表示字符O重复出现指定次数n。因此[O*2]匹配OO的字符串。

大部分tr变种支持字符类和速记控制字符。字符类格式为 [：class]，包含数字、希腊字母、空行、小写、大写、 cntrl键、空格、点记符、图形等等。表 12-1包括最常用的控制字符的速记方式及三位八进制引用方式。

当用一个单字符替换一个字符串或字符范围时，注意字符并不放在方括号里（ []）。一些系统也可以使用方括号，例如可以写成 [“\012”]或“\012”，tr也允许不加引号，因此命令中看到单引号而不是双引号时也不要感到奇怪。

像大多数系统工具一样， tr也受特定字符的影响。因此如果要匹配这些字符，需使用反斜

China-pub.com 下载

线屏蔽其特殊含义。例如，用\{指定花括号左边可以屏蔽其特殊含义。

表12-1 tr中特定控制字符的不同表达方式

速 记 符	含 义	八进制方式
\a	Ctrl-G 铃声	\007
\b	Ctrl-H 退格符	\010
\f	Ctrl-L 走行换页	\014
\n	Ctrl-J 新行	\012
\r	Ctrl-M 回车	\015
\t	Ctrl-I tab键	\011
\v	Ctrl-X	\030

12.1.2 保存输出

要保存输出结果，需将之重定向到一个文件。下面的例子重定向输出到文件 results.txt。输入文件是cops.txt。

```
$ tr -s "[a-z]"< oops.txt >results.txt
```

现在看一些例子。

12.1.3 去除重复出现的字符

下面文件包含了一些打印错误。这种情况时常发生，例如在 vi编辑器中，偶尔按住一个键不放。

```
$ pg oops.txt
And the cowwwwws went homeeeeeeee
Or did theyyyy
```

如果要去除重复字母或将其压缩在一起，使用 -s选项。因为都是字母，故使用[a-z]。输入文件重定向到tr命令。

```
$ tr -s "[a-z]"< oops.txt
And the cows went home
Or did they
```

所有重复字符被压缩成一个。如果使用 cat命令，再将结果管道输出至tr，结果是一样的。

```
$ cat oops.txt | tr -s "[a-z]"
And the cows went home
Or did they
```

12.1.4 删除空行

要删除空行，可将之剔出文件。下面是一个文件 plane.txt。文本间有许多空行。

```
$ pg plane.txt
987932  Spitfire


190992  Lancaster


238991  Typhoon
```

China-pub.com
下载

使用-s来做这项工作。换行的八进制表示为\012，命令为：

```
$ tr -s "[\012]" < plane.txt
987932  Spitfire
190992  Lancaster
238991  Typhoon
```

也可以使用换行速记方式\n，这里用单引号（通常用双引号）。

```
$ tr -s ["\n"] < plane.txt
987932  Spitfire
190992  Lancaster
238991  Typhoon
```

12.1.5 大写到小写

除了删除控制字符，转换大小写是tr最常用的功能。为此需指定即将转换的小写字符[a-z]和转换结果[A-Z]。

第一个例子，tr从一个包含大小写字母的字符串中接受输入。

```
$ echo "May Day, May Day, Going Down.." | tr "[a-z]" "[A-Z]"
MAY DAY, MAY DAY, GOING DOWN..
```

同样，也可以使用字符类[：lower：]和[：upper：]。

```
$ echo "May Day, May Day, Going Down.." | tr "[:lower:]" "[:upper:]"
MAY DAY, MAY DAY, GOING DOWN..
```

将文本文件大写转换为小写并输出至一个新文件，格式为：

cat file-to-translate | tr "[A-Z]" "[a-z]" >new-file-name

这里file-to-translate保存即将转换的文件，new-file-name为保存结果的新文件名。例如：

cat myfile | tr "[A-Z]" "[a-z]" >lower_myfile

12.1.6 小写到大写

转换小写到大写与上一节大写到小写过程刚好相反。以下有两个例子：

```
$ echo " Look for the route, or make the route" | tr "[a-z]" "[A-Z]"
 LOOK FOR THE ROUTE, OR MAKE THE ROUTE

$ echo "May Day, May Day, Going Down.." | tr "[:upper:]" "[:lower:]"
may day, may day, going down..
```

将文本文件从小写转换为大写并将结果存入一个新文件，格式为：

cat file-to-translate | tr "[a-z]" "[A-Z]" >new-file-name

file-to-translate保存即将转换的文件，new-file-name保存结果文件，例如：

cat myfile | tr "[a-z]" "[A-Z]" >upper_myfile

12.1.7 删除指定字符

偶尔会从下载文件中删除只包含字母或数字的列。需要结合使用-c和-s选项完成此功能。

下面的文件包含一个星期的日程表。任务是从其中删除所有数字，只保留日期。日期有大写，也有小写格式。因此需指定两个字符范围[a-z]和[A-Z]，命令tr -cs "[a-z][A-Z]""[\012*]"将

China-pub.com 下载

文件每行所有不包含在[a-z]或[A-Z]（所有希腊字母）的字符串放在字符串1中并转换为一新行。-s选项表明压缩所有新行，-c表明保留所有字母不动。原文件如下，后跟tr命令：

```
$ pg diary.txt
monday 10:50
Tuesday 15:30
wednesday 15:30
thurday 10:30
Friday 09.20
$ tr -cs "[a-z][A-Z]" "[\012*]" < diary.txt
monday
Tuesday
wednesday
thurday
Friday
```

12.1.8 转换控制字符

tr的第一个功能就是转换控制字符，特别是从dos向UNIX下载文件时，忘记设置ftp关于回车换行转换的选项时更是如此。

下面是故意没有设置转换开关的一个文本文件，是关于文具需求的一部分内容。使用 cat -v显示控制字符。

```
$ cat -v stat.tr
Boxes paper^^^^^^12^M
Clips metal^^^^^^50^M
Pencils-meduim^^^^^^10^M
^Z
```

猜想‘^^^^^^’是tab键。每一行以Ctrl-M结尾，文件结尾Ctrl-Z，以下是改动方法。

使用-s选项，查看ASCII表。^的八进制代码是136，^M是015，tab键是011，^Z是032,下面将按步骤完成最终功能。

用tab键替换^^^^^^，命令为"\136""[\011*]"。将结果重定向到临时工作文件stat.tmp。

```
$ tr -s "[\136]" "[\011*]" < stat.tr  >stat.tmp
Boxes paper 12^M
Clips metal 50^M
Pencils-medium 10^M
^Z
```

用新行替换每行末尾的^M，并用\n去除^Z，输入要来自于临时工作文件stat.tmp。

```
$ tr -s "[\015\032]" "\n" < stat.tmp
Boxes paper     12
Clips metal     50
Pencils-medium  10
```

最后去除所有的控制字符，文件就可以使用了。

12.1.9 快速转换

如果需要删除文件中^M，并代之以换行。使用命令：

```
$ tr -s "[\015]" "\n" < input_file
```

或者用下述命令得同样结果。

```
$ tr -s "[\r]" "[\n]" < input_file
```

也可以用下述命令：

China-pub.com 下载


```
$ tr -s "\r" "\n" < input_file
```

另一个一般的Dos到UNIX转换是命令：

```
$ tr -s "[\015\032]" "[\012*]" < input_file
```

将删除所有^M和^Z，代之以换行。

要删除所有的tab键，代之以空格，使用命令：

```
$ tr -s "[\011]" "[\040*]" < input_file
```

替换passwd文件中所有冒号，代之以tab键，可以增加可读性。将冒号引起来，指定替换字符串中tab键八进制值011，下面是passwd文件，后跟tr命令结果：

```
$ pg passwd
halt:*:7:0:halt:/sbin:/sbin/halt
mail:*:8:12:mail:/var/spool/mail:
news:*:9:13:news:/var/spool/news:
uucp:*:10:14:uucp:/var/spool/uucp:

$ tr -s "[:]" "[\011]" < passwd
halt    *    7    0    halt    /sbin    /sbin/halt
mail    *    8    12   mail    /var/spool/mail
news    *    9    13   news    /var/spool/news
uucp    *    10   14   uucp    /var/spool/uucp
```

或者用下述命令得同样结果。这里使用 tab键的速记符。

```
$ tr "[:]" "[\t]" < passwd
```

12.1.10 匹配多于一个字符

可以使用[character*n]格式匹配多于一个字符。下述文件列出系统硬盘信息，其中包含了系统已经注册的和未识别的。第一列是数字，如果不全是 0，表明第二列相应硬盘已经注册。

有时全部为0看起来很烦人，找个吸引人注意力的符号来代替它，以便一眼就能看出哪个硬盘已注册，哪个不可识别。原文件如下：

```
$ pg hdisk.txt
1293 hdisk3
4512 hdisk12
0000 hdisk5
4993 hdisk12
2994 hdisk7
```

从文件列表中知道，有一个硬盘未注册，因此用星号代替所有的 0。模式为[0*4]，意即匹配至少4个0，替换字符串为星号，过滤命令及结果如下：

```
$ tr "[0*4]" "*" <hdisk.txt
1293 hdisk3
4512 hdisk12
**** hdisk5
4993 hdisk12
2994 hdisk7
```

现在从文件中可以直接看出哪个未注册。

12.2 小结

tr主要用于字符转换或者抽取控制字符。本章所有功能都可以用 sed来完成，但有些人宁愿使用tr，因为tr更加快捷、容易。

China-pub.com
下载

第三部分　登 录 环 境

第13章　登 录 环 境

登录系统时，在进入命令提示符前，系统要做两个工作。键入用户名和密码后，系统检查是否为有效用户，为此需查询 /etc/passwd文件。如果登录名正确并且密码有效，开始下一步过程，即登录环境。

本章内容有：

• 登录过程。
• 文件/etc/passwd。
• $HOME.profile。
• 定制$HOME.profile。

在进行下一步处理之前，先看看文件 /etc/passwd。这是一个文本文件，可以任意修改其中的文本域，但要小心。此文本有 7个域，并用冒号作分隔符，以下是其部分文件内容列表。在顶端加有列号，这样各域标识得更加清晰。

```
[ 1 ][          2          ][ 3 ][ 4 ][     5     ][          6          ][  7  ]
kvp:JFqMmk9.uRioA:405:413:K.V.Pally:/home/sysdev/kvp:/bin/sh
dhw:hi/G4U1CUd9aI,B/0J:407:401:D.Whitely:/home/dept47/dhw:/bin/sh
aec:ILgHtxJ9kXtSc,B/GI:408:401:A.E.Cloody:/b_user/dept47/aec:/bin/sh
gdw:iLFu9BB8RNjpc,B/MK:409:401:G.D.Wilcom:/b_user/dept47/gdw:/bin/sh
```

现在来看看各域，第 1域是登录名，第 2域是加密的密码，第 5域是用户全名。第 6域是用户根目录，第7域是用户使用的shell。这里/bin/sh意即缺省为常规Bourne Shell。

Passwd文件可能还有其他格式。其中的一个版本即为实际 passwd域保存在另一个文件中。以上即为最普通格式。

登录成功后，系统执行两个环境设置文件，第一个是 /etc/profile，第二个是 .profile，位于用户根目录下。

系统还会处理其他的初始化文件。这里只涉及 profile文件。

13.1　/etc/profile

用户登录时，自动读取/etc目录下profile文件，此文件包含：

• 全局或局部环境变量。
• PATH信息。
• 终端设置。
• 安全命令。
• 日期信息或放弃操作信息。

下面就来详细解释上述各项内容。设置全局环境变量便于用户及其进程和应用访问它。

China-pub.com 下载

PATH定位包含可执行文件，库文件及一般文本文件的目录位置，便于用户快速访问。终端设置使系统获知用户终端的一般特性。安全命令包括文件创建模式或敏感区域的双登录提示。日期信息是一个文本文件，保存用户登录时即将发生事件的记录或放弃登录的信息文件。

以下是文件/etc/profile，列表后将予以讨论。

```
$ pg /etc/profile
#!/bin/sh
#
trap "" 2 3
#  Set LOGNAME
export LOGNAME

# set additional MAN paths.
MANPATH=/usr/opt/sybase/man
export MANPATH

#  Set TZ.
if [ -f /etc/TIMEZONE ]
then
.  /etc/TIMEZONE
fi

#  Set TERM.
if [ -z "$TERM" ]
then
  TERM=vt220    # for standard async terminal/console
  export TERM
fi

  #  Allow the user to break the Message-Of-The-Day only.
  trap "trap "2"  2
  if [ -f /usr/bin/cat ] ; then
    cat -s /etc/motd
  fi
  trap "" 2
  if [ -f /usr/bin/mail ] ; then
    if mail -e ; then
      echo "Hey guess what? you have mail"
    fi
  fi
  ;;
esac

#  set the umask for more secure operation
umask 022
fi
#  set environments
SYSHOME=/appdvb/menus
ASLBIN=/asl_b/bin
UDTHOME=/dbms_b/ud
UDTBIN=/dbms_b/ud/bin
PAGER=pg
NOCHKLPREQ=1
PATH=$PATH:$UDBIN:$ASLBIN
```

China-pub.com
下载


```
export PATH UDTHOME UDTBIN PAGER NOCHKLPREQ SYSHOME

trap  2 3
#  Set variable SAVEDSTTY so that it can be used to recover the
#  stty settings on coming out of the audit system.
SAVEDSTTY='stty -g'
export SAVEDSTTY

# log all connections to syslog
logger -p local7.info -t login  $LOGNAME `tty`
trap 'logger -p local7.info -t logout $LOGNAME `tty`' 0

# suppress creation of core dumps
ulimit -c 0
#
# check if users are logged in more than twice apart from...
case $LOGNAME in
idink | psalon | dave)
    ;;
  *)
    PID=${$};export PID
    Connected=`who | awk '{print $1}' | fgrep -xc ${LOGNAME}`
    if [ "$Connected" -gt  2 ]
    then
      echo
      echo 'You are logged in more than twice.'
      echo
      who -u | grep $LOGNAME
      echo
      echo 'Enter <CR> to exit \c'
      read FRED
      kill -15 $PID
    fi
    ;;
esac
# set the prompt to hold the user id
PS1="$LOGNAME >"
```

其中一些命令可能不好理解，不必担心，本书以后将陆续予以介绍。如果愿意，可以参照这个列表建立自己的profile文件。

第一行捕获两个信号，即使用QUIT退出用户或<Ctrl-c>键停止文件执行。

接下来导出LOGNAME；然后指定系统额外增加的man页查询的位置。MANPATH将此位置加入存在的man页查询列表中。

检查时区文件，如果存在，指定它作为时区源，设置终端类型为 vt220。

重新设置捕获信号，以便于用户读取日期文件信息，但此后必须再重新设置它。

建立邮件信息（当有新邮件到达时显示此信息）。

设置umask值，使文件创建时带有一定的缺省权限位集。

初始化环境变量，设置路径并导出，以便于用户使用。

重新设置捕获信号<Ctrl-C>和QUIT。

保存缺省的stty设置，便于用户退出查询系统时能够重新初始化终端设置。

China-pub.com 下载

将所有连接注册到文件/var/adm/messages，即缺省系统注册文件中。

使用ulimit命令限制内存溢出或十六进制溢出数目。

下面的一小段脚本限制用户最多同时登录两次，但不包括三个人（idnk，psalom，dave），如果有人试图登录超过两次，则令其退出登录进程。

最后设置命令提示符到登录名。

此环境设置为全局使用，下面在用户自己的profile文件中定制环境。

13.2 用户的$HOME.profile

/etc/profile文件执行时，用户将被放入到自己的$HOME目录中，回过头来观察passwd文件，用户的$HOME目录在倒数第2列。

可以将之看作用户根目录，因为正是在这里存储了所有的私有信息。

如果.profile已经存在，系统将参照此文件，意即对此过程并不创建另一个 shell，因而在/etc/profile下设置的环境不做改动，除非在.profile中强制改动它。如果创建另一个进程，用户本地的shell变量将被覆盖。

回到.profile，一般来说创建帐户时，一个profile文件的基本框架即随之创建。不要忘了在.profile文件中可以通过设置相关条目以不同的值或使用uset命令来覆盖/etc/profile文件中的设置。如果愿意，可以定制用户自己的.profile文件。先来看看标准的.profile文件。

```
$ pg .profile
#.profile
set -a
MAIL=/usr/mail/${LOGNAME:?}
PATH=$PATH
export PATH
#
```

现在改动此文件。

现在加入两个环境变量，如EDITOR，以使cron或其他应用获知正在使用的编辑器；将TERM变量设置为vt100，而不是vt220。

也可以创建bin目录，将之加入路径（path），目录结构中加一个bin目录是一个好习惯。在这里可以保存所有脚本，将之加入PATH后，就不必写入脚本的文件路径名全称，只键入脚本名即可。

几乎没有人想在命令提示符中显示自己的登录名，而宁愿使用现在的目录路径或是正在使用的系统主机名做提示符。例如，下面显示了在命令提示符中如何设置主机名：

```
$ PS1="`hostname`>"
dns-server>
```

如果用户位于当前目录下：

```
$ PS1="\`pwd\`>"
/home/dave>
```

如果上面的命令返回pwd，可使用如下命令：

```
PS1='$PWD >';
```

我通常设置辅助命令提示符（一般用于命令提示符里的多行命令）为符号©，它的ASCII代码值八进制数为251，十进制为169。

China-pub.com
下载


```
$ PS2="`echo "\251"`:"
/home/dave> while read line
©:do
©:echo $line
©:done
```

如果是LINUX，那么……

在echo命令中使用八进制值，方法为：

```
$ PS2="`echo -e "\251"`:"
```

如果需要访问管理区/usr/admin，可将之加入环境变量，这样可以很容易地进入此目录。

```
ADMIN=/usr/adm
```

如果要知道用户本身登录后系统用户数，使用 who和wc命令。

```
$ echo "`who|wc -l` users are on today"

19 users are on today.
```

将上述设置加入.profile文件。如果要使.profile或/etc/profile文件改动生效，可以退出登录然后再登入，或者参照此文件设置。要参照此文件设置，格式为：

```
./pathname/filename
```

要参照.profile设置，键入：

```
$. .profile
```

如果未成功，试试：

```
$. ./profile
```

以下为改动过的.profile文件。

```
$ pg .profile
#.profile
MAIL=/usr/mail/${LOGNAME:?}
PATH=$PATH:$HOME:bin
#
EDITOR=vi
TERM vt100
ADMIN=/usr/adm
PS1="`hostname`>"
PS2="`echo "\0251"`:"
export EDITOR TERM ADMIN PATH PS1
echo "`who|wc -l` users are on to-day"
```

13.3 stty用法

stty用于设置终端特性。要查询现在的stty选项，使用stty -a。

```
$ stty -a
speed 9600 baud; rows 24; columns 80; line = 0;
intr =^C; quit =^\; erase =^?; kill =^U; eof =^D; eol =<undef>;
eol2 =<undef>; start =^Q; stop =^S; susp =^Z; rprnt =^R; werase = ^W;
lnext =^V; flush =^O; min =1; time =0;
-parenb -parodd cs8 -hupcl -cstopb cread -clocal -crtscts
-ignbrk -brkint -ignpar -parmrk -inpck -istrip -inlcr -igncr icrnl ixon
-ixoff -iuclc -ixany -imaxbel
opost -olcuc -ocrnl onlcr -onocr -onlret -ofill -ofdel nl0 cr0 tab0 bs0
vt0 ff0 isig icanon iexten echo echoe echok -echonl -noflsh -xcase
```

China-pub.com 下载

-tostop -echoprt echoctl echoke

设置终端时遇到的一个最普遍问题是退格键不起作用。这不是不可挽救的。本机 stty命令中^?即为退格键，使用<Ctrl-H>可能会退格并删除前一个字符。在命令行中设置一个 stty选项，一般格式为：

```
stty name character
```

以下将退格设置为^H：

```
$ stty erase '\^H'
```

在.profile文件中使用上述命令可能会碰到一些问题，因为 stty期望输入一个实际 'Control H'序列，在vi编辑器环境下使用下述方法解决它：

按住Ctrl键，同时按下V键，释放V键，再按下H键。

最常用的stty命令使用在下述设置上：

名　称	键	含　义
intr	^C	终止进程
echo		打开echo功能
-echo		关闭echo功能
eof	^D	文件尾；注销
kill	^Y	删除一行
start	^Q	滚动屏幕文本
stop	^S	停止滚动屏幕文本

stty的一个可用选项为：

```
stty -g
```

此选项允许以可读格式保存 stty现有设置，便于以后重置回 stty。正像前面在文件/etc/profile中看到的一样。将stty -g内容放入一个变量中，工作完成后，任何改动的设置将被写回stty。

在改变stty设置值并和终端打交道时，此方法很有用。这样可以很容易地存储其初始设置。下面的例子将stty的现有设置保存。使用 stty -g 关掉echo，然后在脚本结尾处保存 stty初始设置。

```
$ pg password
#!/bin/sh
# password
# show use of the restoring stty environment
SAVEDSTTY=`stty -g`
stty -echo
echo "\nGive me that password :\c"
read PASSWD
echo "\nyour password is $PASSWD"
stty $SAVEDSTTY

$ sttypass
Give me that password :
your password is bong
```

如果是LINUX，那么……

要使LINUX知道正在使用字符串中转义字符，echo命令应加入-e，即echo -e。

China-pub.com
下载

（续）

```
SAVEDSTTY=`stty -g`
stty -echo
echo "\nGive me that password :\c"
read PASSWD
echo "\nyour password is $PASSWD"
stty $SAVEDSTTY
```

stty命令可以与终端、打印机、调制解调器打交道，功能十分丰富。使用 stty时要慎重，不要使用已经使用的键或无效值。

13.4 创建.logout文件

使用Bourne shell与其他shell不同，其缺点是不包含.logout文件。此文件保存有执行exit命令时，在进程终止前执行的命令。

但是通过使用trap命令（trap和信号将在本书后面讨论），Bourne shell也可以创建自己的.logout文件。方法如下：编辑.profile文件，在最后一行加入下列命令，然后保存并退出。

```
trap "$HOME /.logout"0
```

再键入一个.logout文件，敲入下列执行命令。如果愿意，可以在此脚本中加入任何命令。

```
$ pg .logout
rm -f $HOME/*.log
rm -f $HOME/*.tmp
echo "Bye...bye $LOGNAME"
```

用户退出时，调用.logout文件。过程如下：用户退出一个shell时，传送了一个信号0，意即从现在shell中退出，在控制返回shell继续退出命令前，.profile文件中trap行将捕获此信号并执行.logout。

13.5 小结

可以定制用户本身的.profile以满足需求，本章讲述了如何覆盖系统设置以满足用户需求。从显示友好信息到终端特性设置，定制用户环境可以有许多方式。

China-pub.com
下载

第14章　环境和shell变量

为使shell编程更有效，系统提供了一些 shell变量。shell变量可以保存诸如路径名、文件名或者一个数字这样的变量名。 shell将其中任何设置都看做文本字符串。

有两种变量，本地和环境。严格地说可以有 4种，但其余两种是只读的，可以认为是特殊变量，它用于向shell脚本传递参数。

本章内容有：

- shell变量。
- 环境变量。
- 变量替换。
- 导出变量。
- 特定变量。
- 向脚本传递信息。
- 在系统命令行下使用位置参数。

14.1　什么是shell变量

变量可以定制用户本身的工作环境。使用变量可以保存有用信息，使系统获知用户相关设置。变量也用于保存暂时信息。例如：一变量为 EDITOR，系统中有许多编辑工具，但哪一个适用于系统呢？将此编辑器名称赋给 EDITOR，这样，在使用 cron或其他需要编辑器的应用时，这就是你将一直使用的EDITOR取值，并将之用作缺省编辑器。

下面是一个例子，登录的审核系统需要编辑。在菜单中选择此选项时，应用查询 EDITOR变量值，其值为vi。系统知道可使用此编辑器。

另一个例子需要登录数据库系统，键入下列命令：

```
$ isql -Udavet -Pabcd -Smethsys
```

这里-S为正在连接的服务器名称。有一变量 DSQUERY保存服务器名称值。设置服务器名称值到DSQUERY变量，这样如果登录时不使用 -S提供服务器名称，应用将查询 DSQUERY变量，并使用其取值作为服务器名称。需要做的全部工作就是键入下列命令：

```
$ isql -Udavet -Pabcd
```

工作方式同上例。

14.2　本地变量

本地变量在用户现在的 shell生命期的脚本中使用。例如，本地变量　file-name取值为loop.doc，这个值只在用户当前shell生命期有意义。如果在shell中启动另一个进程或退出，此值将无效。这个方法的优点就是用户不能对其他的 shell或进程设置此变量有效。

表14-1列出各种实际变量模式

使用变量时，如果用花括号将之括起来，可以防止 shell误解变量值，尽管不必一定要这

样做，但这确实可用。

要设置一本地变量，格式为：

```
$ variable-name=value 或 ${variable-name=value}
```

注意，等号两边可以有空格。如果取值包含空格，必须用双引号括起来。 shell变量可以用大小写字母。

表14-1 变量设置时的不同模式

Variable-name=value	设置实际值到variable-name
Variable-name+value	如果设置了variable-name，则重设其值
Variable-name:?value	如果未设置variable-name，显示未定义用户错误信息
Variable-name?value	如果未设置variable-name，显示系统错误信息
Variable-name:=value	如果未设置variable-name，设置其值
Variable-name:-value	同上，但是取值并不设置到variable-name，可以被替换

14.2.1 显示变量

使用echo命令可以显示单个变量取值，并在变量名前加 $，例如：

```
$ GREAT_PICTURE="die hard"
$ echo ${GREAT_PICTURE}
die hard

$ DOLLAR=99
$ echo ${DOLLAR}
99

$ LAST_FILE=ZLPSO.txt
$ echo ${LAST_FILE}
ZLPSO.txt
```

可以结合使用变量，下面将错误信息和环境变量 LOGNAME设置到变量error-msg。

```
$ ERROR_MSG=" Sorry this file does not exist user $LOGNAME"
$ echo ${ERROR_MSG}
Sorry this file does not exist user dave
```

上面例子中， shell首先显示文本，然后查找变量 $LOGNAME，最后扩展变量以显示整个变量值。

14.2.2 清除变量

使用unset命令清除变量。

```
unset variable-name
```

```
$ PC=enterprise
$ echo ${PC}
enterprise
$ unset PC
$ echo ${PC}
$
```

14.2.3 显示所有本地shell变量

使用set命令显示所有本地定义的shell变量。

China-pub.com 下载

```
$ set
...
PWD=/root
SHELL=/bin/sh
SHLVL=1
TERM=vt100
UID=7
USER=dave
dollar=99
great_picture=die hard
last_file=ZLPSO.txt
```

set输出可能很长。查看输出时可以看出 shell已经设置了一些用户变量以使工作环境更加容易使用。

14.2.4 结合变量值

将变量并排可以使变量结合在一起：

echo ${variable_name}${variable_name}...

```
$ FIRST="Bruce "
$ SURNAME=Willis
$ echo ${FIRST}${SURNAME}
Bruce Willis
```

14.2.5 测试变量是否已经设置

有时要测试是否已设置或初始化变量。如果未设置或初始化，就可以使用另一值。此命令格式为：

```
${variable:-value}
```

意即如果设置了变量值，则使用它，如果未设置，则取新值。例如：

```
$ COLOUR=blue
$ echo "The sky is ${COLOUR:-grey} today"
The sky is blue today
```

变量colour取值blue，echo打印变量colour时，首先查看其是否已赋值，如果查到，则使用该值。现在清除该值，再来看看结果。

```
$ COLOUR=blue
$ unset COLOUR
$ echo "The sky is ${COLOUR:-grey} today"
The sky is grey today
```

上面的例子并没有将实际值传给变量，需使用下述命令完成此功能：

```
${variable:=value}
```

下面是一个更实用的例子。查询工资清单应用的运行时间及清单类型。在运行时间及类型输入时，敲回车键表明用户并没有设置两个变量值，将使用缺省值（ 03:00和Weekly），并传入at命令中以按时启动作业。

```
$ pg vartest
#!/bin/sh
# vartest
echo  "what time do you wish to start the payroll [03:00]:"
read TIME
```

```
echo " process to start at ${TIME:=03:00} OK"
echo  "Is it a monthly or weekly run [Weekly]:"
read RUN_TYPE
echo "Run type is ${RUN_TYPE:=Weekly}"
at -f $RUN_TYPE $TIME
```

在输入域敲回车键，输出结果如下：

```
$ vartest
what time do you wish to start the payroll [03:00]:
 process to start at 03:00 OK
Is it a monthly or weekly run [Weekly]:
Run type is Weekly

warning: commands will be executed using /bin/sh
job 15 at 1999-05-14 03:00
```

也可以编写脚本测试变量是否取值，然后返回带有系统错误信息的结果。下面的例子测试变量file是否取值。

```
$ echo "The file is ${FILES:?}"
sh: files: parameter null or not set
```

以上结果可读性不好，但是可以加入自己的脚本以增加可读性。

```
$ echo "The file is ${FILES:?" sorry cannot locate the variable files"}"

sh: files:  sorry cannot locate the variable files
```

测试变量是否取值，如果未设置，则返回一空串。方法如下：

```
${variable:+value}
```

使用下述方法初始化变量为空字符串。

```
variable=""
$DETINATION=""
```

14.2.6　使用变量来保存系统命令参数

可以用变量保存系统命令参数的替换信息。下面的例子使用变量保存文件拷贝的文件名信息。变量source保存passwd文件的路径，dest保存cp命令中文件目标。

```
$ SOURCE="/etc/passwd"
$ DEST="/tmp/passwd.bak"
$ cp ${SOURCE} ${DEST}
```

下面例子中，变量device保存磁带设备路径，然后用于在mt命令中倒带。

```
$ DEVICE="/dev/rmt/0n"
$ mt -f ${DEVICE} rewind
```

14.2.7　设置只读变量

如果设置变量时，不想再改变其值，可以将之设置为只读方式。如果有人包括用户本人想要改变它，则返回错误信息。格式如下：

```
variable-name=value
readonly variable-name
```

下面的例子中，设置变量为系统磁带设备之一的设备路径，将之设为只读，任何改变其

China-pub.com 下载

值的操作将返回错误信息。

```
$ TAPE_DEV="/dev/rmt/0n"
$ echo ${TAPE_DEV}
/dev/rmt/0n
$ readonly TAPE_DEV
$ TAPE_DEV="/dev/rmt/1n"
sh: TAPE_DEV: read-only variable
```

要查看所有只读变量，使用命令 readonly即可。

```
$ readonly
declare -r FILM="Crimson Tide"
declare -ri PPID="1"
declare -r TAPE_DEV="/dev/rmt/0n"
declare -ri UID="0"
```

14.3 环境变量

环境变量用于所有用户进程（经常称为子进程）。登录进程称为父进程。 shell中执行的用户进程均称为子进程。不像本地变量（只用于现在的 shell）环境变量可用于所有子进程，这包括编辑器、脚本和应用。

环境变量可以在命令行中设置，但用户注销时这些值将丢失，因此最好在 .profile文件中定义。系统管理员可能在 /etc/profile文件中已经设置了一些环境变量。将之放入 profile文件意味着每次登录时这些值都将被初始化。

传统上，所有环境变量均为大写。环境变量应用于用户进程前，必须用 export命令导出。环境变量与本地变量设置方式相同。

14.3.1 设置环境变量

```
VARIABLE-NAME=value；export VARIABLE-NAME
```

在两个命令之间是一个分号，也可以这样写：

```
VARIABLE-NAME=value
Export VARIABLE-NAME
```

14.3.2 显示环境变量

显示环境变量与显示本地变量一样，例子如下：

```
$ CONSOLE=tty1; export CONSOLE
$ echo $CONSOLE
tty1

$ MYAPPS=/usr/local/application; export MYAPPS
$ echo $MYAPPS
/usr/local/application
```

使用env命令可以查看所有的环境变量。

```
$ env
HISTSIZE=1000
HOSTNAME=localhost.localdomain
LOGNAME=dave
MAIL=/var/spool/mail/root
TERM=vt100
HOSTTYPE=i386
```

```
PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/bin:/root/bin:
CONSOLE=tty1
HOME=/home/dave
ASD=sdf
SHELL=/bin/sh
PS1=$
USER=dave
...
```

14.3.3 清除环境变量

使用unset命令清除环境变量：

```
$ unset MYAPPS
$ echo $MYAPPS

$
```

14.3.4 嵌入shell变量

Brourne shell有一些预留的环境变量名，这些变量名不能用作其他用途。通常在/etc/profile中建立这些嵌入的环境变量，但也不完全是，这取决于用户自己。以下是嵌入shell变量列表。

1. CDPATH

改变目录路径变量，保留一系列由冒号隔开的路径名，用于cd命令。如果设置了CDPATH，cd一个目录时，首先查找CDPATH，如果CDPATH指明此目录，则此目录成为当前工作目录。例子如下：

```
$ CDPATH=:/home/dave/bin:/usr/local/apps;export CDPATH.
```

如果要

```
$ cd apps
```

cd命令首先在CDPATH中查找目录列表，如果发现apps，则它成为当前工作目录。

2. EXINIT

EXINIT变量保存使用vi编辑器时的初始化选项。例如，调用vi时，要显示行号，且在第10个空格加入tab键，命令为：

```
$ EXINIT='set nu tab=10';export EXINIT
```

3. HOME

HOME目录，通常定位于passwd文件的倒数第2列，用于保存用户自身文件。设置了HOME目录，可以简单使用cd命令进入它。

```
$ HOME=/home/dave; export HOME
$ pwd
$ /usr/local
$ cd
$ pwd
$ /home/dave
```

也可以用

```
$ cd $ HOME
```

4. IFS

China-pub.com 下载

IFS用作shell指定的缺省域分隔符。原理上讲域分隔符可以是任意字符，但缺省通常为空格、新行或tab键。IFS在分隔文件或变量中各域时很有用。下面的例子将 IFS设置为冒号，然后echo PATH变量，给出一个目录分隔开来的可读性很强的路径列表。

```
$ export IFS=:
$ echo $PATH
/sbin /bin /usr/sbin /usr/bin /usr/X11R6/bin /root/bin
```

要设置其返回初始设置：

```
$ IFS=<space><tab>; export IFS
```

这里<space><tab>为空格和tab键。

5. LOGNAME

此变量保存登录名，应该为缺省设置，但如果没有设置，可使用下面命令完成它：

```
$ LOGNAME='whoami'; export LOGNAME
$ echo $LOGNAME
dave
```

6. MAIL

MAIL变量保存邮箱路径名，缺省为 /var/spool/mail/<login name>。shell周期性检查新邮件，如果有了新邮件，在命令行会出现一个提示信息。如果邮箱并不在以上指定位置，可以用MAIL设置。

```
$ MAIL=/usr/mail/dave;export MAIL
```

7. MAILCHECK

MAILCHECK缺省每60s检查新邮件，但如果不想如此频繁检查新邮件，比如设为每 2m，使用命令：

```
$ MAILCHECK=120;export MAILCHECK
```

8. MAILPATH

如果有多个邮箱要用到MAILPATH，此变量设置将覆盖MAIL设置。

```
$ MAILPATH=/var/spool/dave:/var/spool/admin;export MAILPATH
```

上面的例子中，MAIL检测邮箱dave和admin。

9. PATH

PATH变量保存进行命令或脚本查找的目录顺序，正确排列这个次序很重要，可以在执行命令时节省时间。你一定不想在已知命令不存在的目录下去查找它。通常情况，最好首先放在HOME目录下，接下来是从最常用到一般使用到不常用的目录列表次序。如果要在当前工作目录下查询，无论在哪儿，均可以使用句点操作。目录间用冒号分隔，例如：

```
$ PATH=$HOME/bin:.:/bin:/usr/bin;export PATH
```

使用上面的例子首先查找 HOME/bin目录，然后是当前工作目录，然后是 /bin，最后是 /usr/bin。

PATH可以在系统目录下/etc/profile中设置，也可以使用下面方法加入自己的查找目录。

```
$ PATH=$PATH:/$HOME/bin:.:export PATH
```

这里使用了/etc/profile中定义的PATH，并加入$HOME/bin和当前工作目录。一般来说，在查找路径开始使用当前工作目录不是一个好办法，这样很容易被其他用户发现。

10. PS1

基本提示符包含shell提示符，缺省对超级用户为 #，其他为$。可以使用任何符号作提示

China-pub.com 下载

符，以下为两个例子：

```
$ PS1="star trek:"; export PS1
star trek:
$ PS1="->" ; export PS1
->
```

11. PS2

PS2为附属提示符，缺省为符号 > 。PS2用于执行多行命令或超过一行的一个命令。

```
$ PS2="@:"; export PS2
$ for loop in *
@:do
@:echo $loop
...
```

12. SHELL

SHELL变量保存缺省shell，通常在/etc/passwd中已设置，但是如有必要使用另一个 shell，可以用如下方法覆盖当前shell：

```
$ echo $SHELL
/bin/sh
```

13. TERMINFO

终端初始化变量保存终端配置文件的位置。通常在 /usr/lib/terminfo或/usr/share/terminfo

```
$ TERMINFO=/usr/lib/terminfo;export TERMINFO
```

14. TERM

TERM变量保存终端类型。设置TERM使应用获知终端对屏幕和键盘响应的控制序列类型，常用的有vt100、vt200、vt220-8等。

```
$ TERM=vt100;export TERM
```

15. TZ

时区变量保存时区值，只有系统管理员才可以更改此设置。例如：

```
$ echo $TZ
GMT2EDT
```

返回值表明正在使用格林威治标准时间，与 GMT时差为0，并作EDT保存。

14.3.5 其他环境变量

还有一些预留的环境变量。其他系统或命令行应用将用到它们。以下是最常用的一些，注意这些值均未有缺省设置，必须显示说明。

1. EDITOR

设置编辑器，最常用。

```
$ EDITOR=vi;export EDITOR
```

2. PWD

当前目录路径名，用cd命令设置此选项。

3. PAGER

保存屏幕翻页命令，如pg、more，在查看man文本时用到此功能。

```
$ PAGER='pg -f -p%d';export PAGER
```

4. MANPATH

China-pub.com 下载

保存系统上man文本的目录。目录间用冒号分隔。

```
$ MANPATH=/usr/apps/man:/usr/local/man; export MANPATH
```

5. LPDEST或PRINTER

保存缺省打印机名，用于打印作业时指定打印机名。

```
$ LPDEST=hp3si-systems
```

14.3.6 set命令

在$HOME.profile文件中设置环境变量时，还有另一种方法导出这些变量。使用 set命令-a选项，即set -a指明所有变量直接被导出。不要在 /etc/profile中使用这种方法，最好只在自己的$HOME.profile文件中使用。

```
$ pg .profile
#.profile
set -a
MAIL=/usr/mail/${LOGNAME:?}
PATH=$PATH:$HOME:bin
#
EDITOR=vi
TERM vt220
ADMIN=/usr/adm
PS1="`hostname`>"
```

14.3.7 将变量导出到子进程

shell新用户碰到的问题之一是定义的变量如何导出到子进程。前面已经讨论过环境变量的工作方式，现在用脚本实现它，并在脚本中调用另一脚本（这实际上创建了一个子进程）。

以下是两个脚本列表father和child。

father脚本设置变量film，取值为A Few Good Men，并将变量信息返回屏幕，然后调用脚本child，这段脚本显示第一个脚本里的变量 film，然后改变其值为Die Hard，再将其显示在屏幕上，最后控制返回father脚本，再次显示这个变量。

```
$ pg father
#!/bin/sh
# father script.
echo "this is the father"
FILM="A Few Good Men"
echo "I like the film :$FILM"
# call the child script
child
echo "back to father"
echo "and the film is :$FILM"

$ pg child
#!/bin/sh
# child
echo "called from father..i am the child"
echo "film name is :$FILM"
FILM="Die Hard"
echo "changing film to :$FILM"
```

看看脚本显示结果。

```
$ father
this is the father
I like the film :A Few Good Men
called from father..i am the child
film name is :
changing film to :Die Hard
back to father
and the film is :A Few Good Men
```

因为在father中并未导出变量film，因此child脚本不能将film变量返回。

如果在father脚本中加入export命令，以便child脚本知道film变量的取值，这就会工作：

```
pg father
#!/bin/sh
# father script.
echo "this is the father"
FILM="A Few Good Men"
echo "I like the film :$FILM"
# call the child script
# but export variable first
export FILM
child
echo "back to father"
echo "and the film is :$FILM"

$ father2
this is the father
I like the film :A Few Good Men
called from father..i am the child
film name is :A Few Good Men
changing film to :Die Hard
back to father
and the film is :A Few Good Men
```

因为在脚本中加入了export命令，因此可以在任意多的脚本中使用变量 film，它们均继承了film的所有权。

不可以将变量从子进程导出到文进程， 然面通过重定向就可做到这一点

14.4 位置变量参数

本章开始提到有4种变量，本地、环境，还有两种变量被认为是特殊变量，因为它们是只读的。这两种变量即为位置变量和特定变量参数。先来看一看位置变量。

如果要向一个shell脚本传递信息，可以使用位置参数完成此功能。参数相关数目传入脚本，此数目可以任意多，但只有前 9个可以被访问，使用 shift命令可以改变这个限制。本书后面将讲到shift命令。参数从第一个开始，在第 9个结束；每个访问参数前要加 $符号。第一个参数为0，表示预留保存实际脚本名字。无论脚本是否有参数，此值均可用。

如果向脚本传送Did You See The Full Moon信息，下面的表格讲解了如何访问每一个参数。

$0	$1	$2	$3	$4	$5	$6	$7	$8	$9
脚本名字	Did	You	See	The	Full	Moon			

China-pub.com 下载

14.4.1 在脚本中使用位置参数

在下面脚本中使用上面的例子。

```
$ pg param
#!/bin/sh
# param
echo "This is the script name         : $0"
echo "This is the first parameter     : $1"
echo "This is the second parameter    : $2"
echo "This is the third parameter     : $3"
echo "This is the fourth parameter    : $4"
echo "This is the fifth parameter     : $5"
echo "This is the sixth parameter     : $6"
echo "This is the seventh parameter   : $7"
echo "This is the eighth parameter    : $8"
echo "This is the ninth parameter     : $9"

$ param Did You See The Full Moon
This is the script name        : ./param
This is the first parameter    : Did
This is the second parameter   : You
This is the third parameter    : See
This is the fourth parameter   : The
This is the fifth parameter    : Full
This is the sixth parameter    : Moon
This is the seventh parameter  :
This is the eighth parameter   :
This is the ninth parameter    :
```

这里只传递6个参数，7、8、9参数为空，正像预计的那样。注意，第一个参数表示脚本名，当从脚本中处置错误信息时，此参数有很大作用。

下面的例子返回脚本名称。

```
$ pg param2
#!/bin/sh
echo "Hello world this is $0 calling"

$ param2
Hello world this is ./param2 calling
```

注意$0返回当前目录路径，如果只返回脚本名，在 basename命令下参数设为 $0，刚好得到脚本名字。

```
$ pg param2
#!/bin/sh
echo "Hello world this is `basename $0` calling"

$ param2
Hello world this is param2 calling
```

14.4.2 向系统命令传递参数

可以在脚本中向系统命令传递参数。下面的例子中，在 find命令里，使用 $1参数指定查找文件名。

China-pub.com
下载


```
$ pg findfile
#!/bin/sh
# findfile
find / -name $1 -print

$ findfile passwd
/etc/passwd
/etc/uucp/passwd
/usr/bin/passwd
```

另一个例子中，以 $1向grep传递一个用户 id号，grep使用此id号在passwd中查找用户全名。

```
$ pg who_is
#!/bin/sh
# who_is
grep $1 passwd | awk -F: {print $4}'

$ who_is seany
Seany Post
```

14.4.3 特定变量参数

既然已经知道了如何访问和使用 shell脚本中的参数，多知道一点相关信息也是很有用的，有必要知道脚本运行时的一些相关控制信息，这就是特定变量的由来。共有 7个特定变量，见表14-2。

表14-2 特定shell变量

$#	传递到脚本的参数个数
$*	以一个单字符串显示所有向脚本传递的参数。与位置变量不同，此选项参数可超过 9个
$$	脚本运行的当前进程ID号
$!	后台运行的最后一个进程的进程ID号
$@	与$#相同，但是使用时加引号，并在引号中返回每个参数
$-	显示shell使用的当前选项，与set命令功能相同
$?	显示最后命令的退出状态。0表示没有错误，其他任何值表明有错误。

现在来修改脚本param并替换各种特定变量，与以前的例子不同，用不同的传递文本重新运行脚本。

```
$ pg param
#!/bin/sh
# allparams
echo "This is the script name          : $0"
echo "This is the first parameter      : $1"
echo "This is the second parameter     : $2"
echo "This is the third parameter      : $3"
echo "This is the fourth parameter     : $4"
echo "This is the fifth parameter      : $5"
echo "This is the sixth parameter      : $6"
echo "This is the seventh parameter    : $7"
echo "This is the eighth parameter     : $8"
echo "This is the ninth parameter      : $9"
echo "The number of arguments passed   :$œ"
echo "Show all arguments               : $*"
```

China-pub.com 下载

```
echo "Show me my process ID             : $$"
echo "Show me the arguments in quotes   :" "$@"
echo "Did my script go with any errors  :$?"
```

```
$ param Merry Christmas Mr Lawrence
This is the script name          :./param
This is the first parameter      : Merry
This is the second parameter     : Christmas
This is the third parameter      : Mr Lawrence
This is the fourth parameter     :
This is the fifth parameter      :
This is the sixth parameter      :
This is the seventh parameter    :
This is the eighth parameter     :
This is the ninth parameter      :
The number of arguments passed   :3
Show all arguments               : Merry Christmas Mr Lawrence
Show me my process ID            : 630
Show me the arguments in quotes  : "Merry" "Christmas" "Mr Lawrence"
Did my script go with any errors :0
```

特定变量的输出使用户获知更多的脚本相关信息。可以检查传递了多少参数，进程相应的ID号，以免我们想杀掉此进程。

14.4.4 最后的退出状态

注意，$?返回0。可以在任何命令或脚本中返回此变量以获得返回信息。基于此信息，可以在脚本中做更进一步的研究，返回0意味着成功，1为出现错误。

下面的例子拷贝文件到/tmp，并使用$?检查结果。

```
$ cp ok.txt /tmp
$ echo $?
0
```

现在尝试将一个文件拷入一个不存在的目录：

```
$ cp ok.txt /usr/local/apps/dsf
cp: cannot create regular file '/usr/local/apps/dsf': No such file or
directory

$ echo $?
1
```

使用$?检验返回状态，可知脚本有错误，但同时发现 cp：cannot...，因此检验最后退出状态已没有必要。在脚本中可以用系统命令处理输出格式，要求命令输出不显示在屏幕上。为此可以将输出重定向到 /dev/null，即系统bin中。现在怎样才能知道脚本正确与否？好，这时可以用最后退出状态命令了。请看上一个例子的此形式的实际操作结果。

```
$ cp ok.txt /usr/local/apps/dsf >/dev/null 2>&1
$ echo $?
1
```

通过将包含错误信息的输出重定向到系统 bin中，不能获知最后命令返回状态，但是通过使用$!,（其返回值为1）可知脚本失败。

检验脚本退出状态时，最好将返回值设置为一个有意义的名字，这样可以增加脚本的可读性。

China-pub.com
下载


```
$ cp ok.txt /usr/local/apps/dsf >/dev/null 2>&1
$ cp_status=$?
$ echo $cp_status
1
```

14.5　小结

变量可以使shell编程更容易。它能够保存输入值并提高效率。 shell变量几乎可以包含任何值。特定变量增强了脚本的功能并提供了传递到脚本的参数的更多信息。

China-pub.com 下载

第15章 引 号

上一章介绍了变量和替换操作，在脚本中执行变量替换时最容易犯的一个错误就是由于引用错误。在命令行中引用是很重要的。

本章内容有：

- 引用的必要性。
- 双引、单引和反引号。
- 使用反斜线实现屏蔽。

15.1 引用必要性

这里只讲述引用的基本规则。因为使用引用的例子很多。本书接下来的两个部分将一一予以讲述。脚本中执行行操作时， shell将对脚本设置予以解释。要采取一种方法防止 shell这样做，即使用引用号，包括各式引用或使用反斜线。

一些用户在对文本字符串进行反馈操作时觉得使用引用很麻烦。有时不注意，只引用了一半，这时问题出现了。最好在反馈文本字符串时使用双引号。下面是各种引用的例子。

```
$ echo Hit the star button to exit *

Hit the star button to exit DIR_COLORS HOSTNAME Muttrc X11 adjtime
aliases alias
...
```

文本返回了，但由于未使用双引号， *被shell误解， shell认为用户要做目录列表。用双引号得结果如下：

```
$ echo "Hit the star button to exit *"
Hit the star button to exit *
```

这样就不会有误解产生。表15-1列出各种引用类型。

表15-1 shell引用类型

""	双引号	`	反引号
''	单引号	\	反斜线

15.2 双引号

使用双引号可引用除字符$、`、\外的任意字符或字符串。这些特殊字符分别为美元符号，反引号和反斜线，对shell来说，它们有特殊意义。如果使用双引号将字符串赋给变量并反馈它，实际上与直接反馈变量并无差别。

```
$ STRING="MAY DAY, MAY DAY, GOING DOWN"
$ echo "$STRING"
MAY DAY, MAY DAY, GOING DOWN

$ echo $STRING
MAY DAY, MAY DAY, GOING DOWN
```

China-pub.com 下载

现在假定要设置系统时间输出到变量mydate。

```
$ MYDATE="date"
$ echo $MYDATE
date
```

因为shell将""符号里的字符串赋予变量mydate，date已没有特定意义，故此变量只保存单词date。

如果要查询包含空格的字符串，经常会用到双引号。以下使用grep抽取名字“ Davey Wire ”，因为没有加双引号，grep将“ Davey ”认作字符串，而把“ Wire ”当作文件名。

```
$ grep "Davey Wire" /etc/passwd
grep: Wire: No such file or directory
```

要解决这个问题，可将字符串加双引号。这样 shell会忽略空格，当使用字符时，应总是使用双引号，无论它是单个字符串或是多个单词。

```
$ grep "Davey Wire" /etc/passwd
davyboy:9sdJUK2s:106:Davey Wire:/home/ap
```

在一个反馈的文本行里可以使用双引号将变量引起来。下面的例子中， shell反馈文本行，遇到符号$，知道这是一个变量，然后用变量值boy替换变量$BOY。

```
$ BOY="boy"
$ echo " The $BOY did well"
 The boy did well

$ echo " The "$BOY" did well"
 The boy did well
```

15.3 单引号

单引号与双引号类似，不同的是shell会忽略任何引用值。换句话说，如果屏蔽了其特殊含义，会将引号里的所有字符，包括引号都作为一个字符串。使用上一个例子，结果如下：

```
$ GIRL='girl'
$ echo "The '$GIRL' did well"
The 'girl' did well
```

15.4 反引号

反引号用于设置系统命令的输出到变量。 shell将反引号中的内容作为一个系统命令，并执行其内容。使用这种方法可以替换输出为一个变量。反引号可以与引号结合使用。下面将举例说明。

下面的例子中， shell试图替代单词hello为系统命令并执行它，因为hello脚本或命令不存在，返回错误信息。

```
$ echo `hello`
sh: hello: command not found
```

现在用date命令再试一次。

```
$ echo `date`
Sun May 16 16:40:19 GMT 1999
```

这次命令有效，shell正确执行。

China-pub.com 下载

下面将命令输出设置为变量mydate，时间格式如下：

```
$ date +%A" the "%e" of "%B" "%Y
Sunday the 16 of May 1999
```

设置到mydate，并显示其值：

```
$ mydate=`date +%A" the "%e" of "%B" "%Y`
$ echo $mydate
Sunday the 16 of May 1999
```

当然也可以将date命令输出至mydate：

```
$ mydate=`date`
$ echo $mydate
Sun May 16 16:48:16 GMT 1999
```

另一个例子中，将反引号嵌在双引号里：

```
$ echo "The date today is `date`"
The date today is Sun May 16 16:56:53 GMT 1999
```

打印当前系统上用户数目：

```
$ echo "There are `who | wc -l` users on the system"
There are  13   users on the system
```

上面的例子中，打印字符串后，shell遇到反引号，将其看作一条命令执行它。

15.5 反斜线

如果下一个字符有特殊含义，反斜线防止shell误解其含义，即屏蔽其特殊含义。下述字符包含有特殊意义：& * + ^ $ ` " | ?。

假定echo命令加*，意即以串行顺序打印当前整个目录列表，而不是一个星号 *。

```
$ echo *


conf.linuxconf conf.modules cron.daily cron.hourly cron.monthly
cron.weekly crontab csh.cshrc default dosemu.conf dosemu.users exports
fdprm fstab gettydefs gpm-root.c
onf group group- host.conf hosts hosts.allow hosts.deny httpd inetd
...
```

为屏蔽星号特定含义，可使用反斜线。

```
$ echo \*
*
```

上述语句同样可用于$$命令，shell解释其为现在进程ID号，使用反斜线屏蔽此意，仅打印$$。

```
$ echo $$
284
$ echo \$$
$$
```

在打印字符串时要加入八进制字符（ASCII相应字符），必须在前面加反斜线，否则shell将其当作普通数字处理。

```
$ echo " This is a copyright 251 sign"
 This is a copyright \251 sign
$ echo  " This is a copyright \251 sign"
 This is a copyright © sign
```

China-pub.com
下载

如果是LINUX，则……

记住使用-e选项来显示控制字符。

```
$ echo -e "This is a copyright \251 sign"
 This is a copyright © sign
```

使用命令expr时，用*表示乘法会出现错误，在*前加上反斜线才会正确。

```
$ expr 12 * 12
expr: syntax error

$ expr 12 \* 12
144
```

在echo命令中加入元字符，必须用反斜线起屏蔽作用。下面的例子要显示价格 $19.99。其中$屏蔽与不屏蔽将产生不同的结果。

```
$ echo "That video looks a good price for $19.99"
That video looks a good price for 9.99
```

使用反斜线屏蔽$，可得更好的结果。

```
$ echo "That video looks a good price for \$19.99"
That video looks a good price for $19.99
```

15.6 小结

在引用时会遇到一些问题且经常出错。我在使用引用时遵循两条规则：

1) 反馈字符串用双引号；但不要引用反馈本身。

2) 如果使用引用得到的结果不理想，再试另一种，毕竟只有三种引用方式，可以充分尝试。

China-pub.com
下载

第四部分 基础shell编程

第16章 shell脚本介绍

一个shell脚本可以包含一个或多个命令。当然可以不必只为了两个命令就编写一个 shell脚本，一切由用户自己决定。

本章内容有：

• 使用shell脚本的原因。
• shell脚本基本元素。
• shell脚本运行方式。

16.1 使用shell脚本的原因

shell脚本在处理自动循环或大的任务方面可节省大量的时间，且功能强大。如果你有处理一个任务的命令清单，不得不一个一个敲进去，然后观察输出结果，再决定它是否正确，如果正确，再继续下一个任务，否则再回到清单一步步观察。一个任务可能是将文件分类、向文件插入文本、迁移文件、从文件中删除行、清除系统过期文件、以及系统一般的管理维护工作等等。创建一个脚本，在使用一系列系统命令的同时，可以使用变量、条件、算术和循环快速创建脚本以完成相应工作。这比在命令行下一个个敲入要节省大量的工作时间。shell脚本可以在行命令中接收信息，并使用它作为另一个命令的输入。

对于不同的UNIX和LINUX，使用一段shell脚本将需要一些小小的改动才能运行通过。实际上shell的可迁移性不成问题，但是系统间命令的可迁移性存在差别。

试试新思路

如果写一段脚本，其执行结果与预想的不同，不必着急。无论多不可思议的结果，记住先把它保存起来，这是修改的基础。这里要说的意思是不要害怕对待新事物，否则将不能树立信心，学起来会更加困难。

16.2 脚本内容

本章不讲怎样设计精巧的脚本，而是怎样使脚本重复利用率高。当通过一些易理解的脚本就可实现同样功能时，没有必要使脚本复杂化。如果作者要写这样一本书，可能会给你留下深刻印象，但这要花费更多的时间和精力去研读和体会脚本。这不是本书的目标。本书脚本流程仅使用基本的脚本技术，十分容易学，然后使用者就可以着手实践了。

脚本不是复杂的程序，它是按行解释的。脚本第一行总是以 #!/bin/sh开始，这段脚本通知shell使用系统上的Bourne shell解释器。

任何脚本都可能有注释，加注释需要此行的第一个字符为 #，解释器对此行不予解释。在

China-pub.com 下载

第二行注释中写入脚本名是一个好习惯。

脚本从上到下执行，运行脚本前需要增加其执行权限。确保正确建立脚本路径，这
用文件名就可以运行它了。

16.3 运行一段脚本

下面是一个已经讨论过的例子，此文件为cleanup。

```
$ pg cleanup
#!/bin/sh
# name: cleanup
# this is a general cleanup script
echo "starting cleanup...wait"
rm /usr/lcoal/apps/log/*.log
tail -40 /var/adm/mesages >/tmp/messages
rm /var/adm/messages
mv /tmp/messages /var/adm/messages
echo "finished cleanup"
```

上述脚本通过将目录下文件名截断，清除/usr/adm/下信息，并删除/usr/local/apps/l
有注册信息。

可以使用chmod命令增加脚本执行权限。

```
$ chmod u+x cleanup
```

现在运行脚本，只敲入文件名即可。

```
$ cleanup
```

如果返回错误信息：

```
$ cleanup
sh:cleanup:command not found
```

再试：

```
$. /cleanup
```

如果脚本运行前必须键入路径名，或者shell结果通知无法找到命令，就需要在
PATH下加入用户可执行程序目录。要确保用户在自己的$HOME可执行程序目录下，应

```
$ pwd
$ /home/dave/bin
```

如果pwd命令最后一部分是bin，那么需要在路径中加入此信息。编辑用户 profi
加入可执行程序目录$HOME/bin如下：

```
PATH=$PATH:$HOME/bin
```

如果没有bin目录，就创建它。首先确保在用户根目录下。

```
$ cd $HOME
$ mkdir bin
```

现在可以在.profile文件中将bin目录加入PATH变量了，然后重新初始化.profile。

```
$. ./profile
```

脚本将会正常运行。

如果还有问题，见第2章和第13章，那里详细介绍了如何解决这一问题。

全书有许多脚本清单，这些脚本都是完整的。将这些脚本输入文件，保存并退

China-pub.com
下载

用chmod命令增加其执行权限，这些脚本就可以实际操作了。

16.4 小结

本章介绍了shell脚本的基本原理，相信关于脚本的功能原理这些已经足够了，读本章时可加快速度。本章目标只是要用户知道运行 shell脚本需要做些什么。

China-pub.com
下载

第17章 条件测试

写脚本时，有时要判断字符串是否相等，可能还要检查文件状态或是数字测试。基于这些测试才能做进一步动作。Test命令用于测试字符串，文件状态和数字，也很适合于下一章将提到的if、then、else条件结构。

本章内容有：

- 对文件、字符串和数字使用test命令。
- 对数字和字符串使用expr命令。

expr命令测试和执行数值输出。使用最后退出状态命令 $?可测知test和expr，二者均以0表示正确，1表示返回错误。

17.1 测试文件状态

test一般有两种格式，即：

```
test condition
```

或

```
[condition]
```

使用方括号时，要注意在条件两边加上空格。

测试文件状态的条件表达式很多，但是最常用的可在表 17-1中查到。

表17-1 文件状态测试

-d	目录	-s	文件长度大于0、非空
-f	正规文件	-w	可写
-L	符号连接	-u	文件有suid位设置
-r	可读	-x	可执行

使用两种方法测试文件scores.txt是否可写并用最后退出状态测试是否成功。记住，0表示成功，其他为失败。

```
$ ls -l scores.txt
-rw-r--r--   1 dave   admin  0 May 15 11:29 scores.txt
$ [ -w scores.txt ]
$ echo $?
0

$ test -w scores.txt
$ echo $?
0
```

两种状态均返回0，可知文件scores.txt可写，现在测试其是否可执行：

```
$ [ -x scores.txt ]
$ echo $?
1
```

查看文件scores.txt权限列表，可知结果正如所料。

下面的例子测试是否存在appsbin目录

China-pub.com
下载


```
drwxr-xr-x  2 dave  admin    1024 May 15 15:53 appsbin
$ [ -d appsbin ]
$ echo $?
0
```

目录appsbin果然存在。

测试文件权限是否设置了suid位

```
-rwsr-xr--  1 root  root    28 Apr 30 13:12 xab
$ [ -u xab ]
$ echo $?
0
```

从结果知道suid位已设置。

17.2 测试时使用逻辑操作符

测试文件状态是否为OK，但是有时要比较两个文件状态。 shell提供三种逻辑操作完成此功能。

-a 逻辑与，操作符两边均为真，结果为真，否则为假。

-o 逻辑或，操作符两边一边为真，结果为真，否则为假。

! 逻辑否，条件为假，结果为真。

下面比较两个文件：

```
-rw-r--r--    1 root    root      0 May 15 11:29 scores.txt
-rwxr-xr--    1 root    root      0 May 15 11:49 results.txt
```

下面的例子测试两个文件是否均可读。

```
$ [ -w results.txt -a -w scores.txt ]
$ echo $?
0
```

结果为真。

要测试其中一个是否可执行，使用逻辑或操作。

```
$ [ -x results.txt -o -x scores.txt ]
$ echo $?
0
```

scores.txt不可执行，但results.txt可执行。

要测试文件results.txt是否可写、可执行：

```
$ [ -w results.txt -a -x results.txt ]
$ echo $?
0
```

结果为真。

17.3 字符串测试

字符串测试是错误捕获很重要的一部分，特别在测试用户输入或比较变量时尤为重要。字符串测试有5种格式。

```
test "string"
test string_operator "string"
test "string" string_operator "string"
[ string_operator string ]
```

China-pub.com 下载

[string string_operator string]

这里，string_operator可为：

= 两个字符串相等。

！= 两个字符串不等。

-z 空串。

-n 非空串。

要测试环境变量EDITOR是否为空：

```
$ [ -z $EDITOR ]
$ echo $?
1
```

非空，取值是否是vi？

```
$ [ $EDITOR = "vi" ]
$ echo $?
0
```

是的，用echo命令反馈其值：

```
$ echo $EDITOR
vi
```

测试变量tape与变量tape2是否相等：

```
$ TAPE="/dev/rmt0"
$ TAPE2="/dev/rmt1"
$ [ "$TAPE" = "$TAPE2" ]
$ echo $?
1
```

不相等。没有规定在设置变量时一定要用双引号，但在进行字符串比较时必须这样做。

测试变量tape与tape2是否不相等。

```
$ [ "$TAPE" != "$TAPE2" ]
$ echo $?
0
```

是的，它们不相等。

17.4 测试数值

测试数值可以使用许多操作符，一般格式如下：

`"number"numeric_operator"number"`

或者

`["number"numeric_operator"number"]`

numeric_operator可为：

-eq 数值相等。

-ne 数值不相等。

-gt 第一个数大于第二个数。

-lt 第一个数小于第二个数。

-le 第一个数小于等于第二个数。

-ge 第一个数大于等于第二个数。

下面的例子返回结果都一样。均为测试两个数是否相等（130是否等于130）。

China-pub.com 下载


```
$ NUMBER=130
$ [ "$NUMBER" -eq "130" ]
$ echo $?
0
```

结果果然正确。

改变第二个数，结果返回失败，状态1（130不等于200）

```
$ [ "$NUMBER" -eq "100" ]
$ echo $?
1
```

测试130是否大于100：

```
$ [ "$NUMBER" -gt "100" ]
$ echo $?
0
```

当然。

也可以测试两个整数变量。下面测试变量 source_count是否小于dest_count:

```
$ SOURCE_COUNT=13
$ DEST_COUNT=15
$ [ "$DEST_COUNT" -gt "$SOURCE_COUNT" ]
$ echo $?
0
```

可以不必将整数值放入变量，直接用数字比较即可，但要加引号。

```
$ [ "990" -le "995" ]
$ echo $?
0
```

可以用逻辑操作符将两个测试表达式结合起来。仅需要用到一对方括号，而不能用两个，否则将返回错误信息“ too many arguments ”。

```
$ [ "990" -le "995" ] -a [ "123" -gt "33" ]
sh:[: too many arguments
```

下面例子测试两个表达式，如果都为真，结果为真，正确使用方式应为：

```
$ [ "990" -le "995"  -a  "123" -gt "33" ]
$ echo $?
0
```

17.5 expr用法

expr命令一般用于整数值，但也可用于字符串。一般格式为：

```
expr argument operator argument
```

expr也是一个手工命令行计数器。

```
$ expr 10 + 10
20
$ expr 900 + 600
1500

$ expr 30 / 3
10

$ expr 30 / 3 / 2
5
```

使用乘号时，必须用反斜线屏蔽其特定含义。因为 shell可能会误解显示星号的意义。

China-pub.com 下载

```
$ expr 30 \* 3
90
```

17.5.1 增量计数

expr在循环中用于增量计算。首先，循环初始化为 0，然后循环值加1，反引号的用法意即替代命令。最基本的一种是从（expr）命令接受输出并将之放入循环变量。

```
$ LOOP=0
$ LOOP=`expr $LOOP + 1`
```

17.5.2 数值测试

可以用expr测试一个数。如果试图计算非整数，将返回错误。

```
$ expr rr + 1
expr: non-numeric argument
```

这里需要将一个值赋予变量（不管其内容如何），进行数值运算，并将输出导入 dev/null，然后测试最后命令状态，如果为0，证明这是一个数，其他则表明为非数值。

```
$ VALUE=12
$ expr $VALUE + 10 > /dev/null 2>&1
$ echo $?
0
```

这是一个数。

```
$ VALUE=hello
$ expr $VALUE + 10 > /dev/null 2>&1
$ echo $?
2
```

这是一个非数值字符。

expr也可以返回其本身的退出状态，不幸的是返回值与系统最后退出命令刚好相反，成功返回1，任何其他值为无效或错误。下面的例子测试两个字符串是否相等，这里字符串为“ hello ”和“ hello ”。

```
$ VALUE=hello
$ expr $VALUE = "hello"
1
$ echo $?
0
```

expr返回1。不要混淆了，这表明成功。现在检验其最后退出状态，返回 0表示测试成功，“ hello ”确实等于“ hello ”。

17.5.3 模式匹配

expr也有模式匹配功能。可以使用 expr通过指定冒号选项计算字符串中字符数。 .*意即任何字符重复0次或多次。

```
$ VALUE=accounts.doc
$ expr $VALUE : October 8, '.*'
12
```

在expr中可以使用字符串匹配操作，这里使用模式 .doc抽取文件附属名。

China-pub.com
下载


```
$ expr $VALUE : '\(.*\).doc'
accounts
```

17.6 小结

本章涉及expr和test基本功能，讲到了怎样进行文件状态测试和字符串赋值，使用其他的条件表达式如if then else和case可以进行更广范围的测试及对测试结果采取一些动作。

China-pub.com
下载

第18章　控制流结构

所有功能脚本必须有能力进行判断，也必须有能力基于一定条件处理相关命令。本章讲述这方面的功能，在脚本中创建和应用控制结构。

本章内容有：

• 退出状态。
• while、for和until loops循环。
• if then else语句。
• 脚本中动作。
• 菜单。

18.1　退出状态

在书写正确脚本前，大概讲一下退出状态。任何命令进行时都将返回一个退出状态。如果要观察其退出状态，使用最后状态命令：

```
$ echo $?
```

主要有4种退出状态。前面已经讲到了两种，即最后命令退出状态 $?和控制次序命令（$$、||）。其余两种是处理shell脚本或shell退出及相应退出状态或函数返回码。在第 19章讲到函数时，也将提到其返回码。

要退出当前进程，shell提供命令exit，一般格式为：

```
exit n
```

其中，n为一数字。

如果只在命令提示符下键入exit，假定没有在当前状态创建另一个shell，将退出当前shell。如果在脚本中键入exit，shell将试图（通常是这样）返回上一个命令返回值。有许多退出脚本值，但其中相对于脚本和一般系统命令最重要的有两种，即：

退出状态0　退出成功，无错误。

退出状态1　退出失败，某处有错误。

可以在shell脚本中加入自己的退出状态（它将退出脚本）。本书鼓励这样做，因为另一个shell脚本或返回函数可能要从 shell脚本中抽取退出脚本。另外，相信加入脚本本身的退出脚本值是一种好的编程习惯。

如果愿意，用户可以在一个用户输入错误后或一个不可覆盖错误后或正常地处理结束后退出脚本。

注意　从现在起，本书所有脚本都将加入注释行。注释行将解释脚本具体含义，帮助用户理解脚本。可以在任何地方加入注释行，因为其本身被解释器忽略。注释行应以#开头。

18.2　控制结构

几乎所有的脚本里都有某种流控制结构，很少有例外。流控制是什么？假定有一个脚本

包含下列几个命令：

```
#!/bin/sh
# make a directory
mkdir /home/dave/mydocs
# copy all doc files
cp *.docs /home/dave/docs
# delete all doc files
rm *.docs
```

上述脚本问题出在哪里？如果目录创建失败或目录创建成功文件拷贝失败，如何处理？这里需要从不同的目录中拷贝不同的文件。必须在命令执行前或最后的命令退出前决定处理方法。shell会提供一系列命令声明语句等补救措施来帮助你在命令成功或失败时，或需要处理一个命令清单时采取正确的动作。

这些命令语句大概分两类：

循环和流控制。

18.2.1 流控制

if、then、else语句提供条件测试。测试可以基于各种条件。例如文件的权限、长度、数值或字符串的比较。这些测试返回值或者为真（0），或者为假（1）。基于此结果，可以进行相关操作。在讲到条件测试时已经涉及了一些测试语法。

case语句允许匹配模式、单词或值。一旦模式或值匹配，就可以基于这个匹配条件作其他声明。

18.2.2 循环

循环或跳转是一系列命令的重复执行过程，本书提到了3种循环语句：

for 循环　每次处理依次列表内信息，直至循环耗尽。

Until 循环　此循环语句不常使用，until循环直至条件为真。条件部分在循环末尾部分。

While 循环　while循环当条件为真时，循环执行，条件部分在循环头。

流控制语句的任何循环均可嵌套使用，例如可以在一个for循环中嵌入另一个for循环。

现在开始讲解循环和控制流，并举一些脚本实例。

从现在起，脚本中echo语句使用LINUX或BSD版本，也就是说使用echo方法echo -e -n，意即从echo结尾中下一行执行命令。应用于UNIX（系统V和BSD）的统一的echo命令参阅19章shell函数。

18.3 if then else语句

if语句测试条件，测试条件返回真（0）或假（1）后，可相应执行一系列语句。if语句结构对错误检查非常有用。其格式为：

```
if 条件1
   then 命令1
  elif 条件2
then 命令2
else 命令3
```

China-pub.com 下载

fi

让我们来具体讲解if语句的各部分功能。

If 条件1 如果条件1为真
Then 那么
  命令1 执行命令1
elif 条件2 如果条件1不成立
then 那么
命令2 执行命令2
else 如果条件1，2均不成立
  命令3 那么执行命令3
fi 完成

if语句必须以单词fi终止。在if语句中漏写fi是最一般的错误。我自己有时也是这样。

elif和else为可选项，如果语句中没有否则部分，那么就不需要 elif和else部分。If语句可以有许多elif部分。最常用的if语句是if then fi结构。

下面看一些例子。

18.3.1 简单的if语句

最普通的if语句是：

if条件
then 命令
fi

使用if语句时，必须将then部分放在新行，否则会产生错误。如果要不分行，必须使用命令分隔符。本书其余部分将采取这种形式。现在简单 if语句变为：

if 条件；then
  命令
fi

注意，语句可以不这样缩排，但建议这样做，因为可以增强脚本的清晰程度。在条件流下采取命令操作更方便。下面的例子测试 10是否小于12，此条件当然为真。因为条件为真， if语句内部继续执行，这里只有一个简单的 echo命令。如果条件为假，脚本退出，因为此语句无else部分。

```
$ pg iftest
#!/bin/sh
# iftest
# this is a comment line, all comment lines start with a #
if [ "10" -lt "12" ]
then
  # yes 10 is less than 12
  echo "Yes, 10 is less than 12"
fi
```

18.3.2 变量值测试

通过测试设置为接受用户输入的变量可以测知用户是否输入信息。下面的例子中测试用

China-pub.com 下载

户键入return键后变量name是否包含任何信息。

```
$ pg iftest2
#!/bin/sh
# if test2
echo -n "Enter your name :"
read NAME
# did the user just hit return ????
if [ "$NAME" = "" ] ; then
  echo "You did not enter any information"
fi
$ iftest2
Enter your name :
You did not enter any information
```

18.3.3 grep输出检查

不必拘泥于变量或数值测试，也可以测知系统命令是否成功返回。对 grep使用if语句找出grep是否成功返回信息。下面的例子中 grep用于查看Dave是否在数据文件 data.file中，注意‘Dave\>’用于精确匹配。

```
$ pg grepif
#!/bin/sh
# grepif
if grep 'Dave\>' data.file > /dev/null 2>&1
then
  echo "Great Dave is in the file"
else
  echo "No Dave is not in the file"
fi

$ grepif
No Dave is not in the file
```

上面的例子中，grep输出定向到系统垃圾堆。如果匹配成功，grep返回0，将grep嵌入if语句；如果grep成功返回，if部分为真。

18.3.4 用变量测试grep输出

正像前面看到的，可以用 grep作字符串操作。下面的脚本中，用户输入一个名字列表，grep在变量中查找，要求其包含人名 Peter。

```
$ pg grepstr
#!/bin/sh
# grepstr
echo -n "Enter a list of names:"
read list
if echo $list | grep "Peter" > /dev/null 2>&1
then
  echo "Peter is here"
  # could do some processing here..
else
  echo "Peter's not in the list. No comment!"
fi
```

以下是对应输入名称的输出信息。

China-pub.com 下载

```
$ grepstr
Enter a list of names:John Louise Peter James
Peter is here
```

18.3.5 文件拷贝输出检查

下面测试文件拷贝是否正常，如果 cp命令并没有拷贝文件 myfile到myfile.bak，则打印错误信息。注意错误信息中`basename $0`打印脚本名。

如果脚本错误退出，一个好习惯是显示脚本名并将之定向到标准错误中。用户应该知道产生错误的脚本名。

```
$ pg ifcp
#!/bin/sh
# ifcp
if cp myfile myfile.bak; then
  echo "good copy"
else
  echo "`basename $0`: error could not copy the files" >&2
fi

$ ifcp
cp: myfile: No such file or directory
ifcp: error could not copy the files
```

注意，文件可能没找到，系统也产生本身的错误信息，这类错误信息可能与输出混在一起。既然已经显示系统错误信息获知脚本失败，就没必要显示两次。要去除系统产生的错误和系统输出，只需简单的将标准错误和输出重定向即可。修改脚本为： >/dev/null 2>&1。

```
$ pg ifcp
#!/bin/sh
# ifcp
if cp myfile myfile.bak >/dev/null 2> then
  echo "good copy"
else
  echo "`basename $0`: error could not copy the files" >&2
fi
```

脚本运行时，所有输出包括错误重定向至系统垃圾堆。

```
$ ifcp
ifcp: error could not copy the files
```

18.3.6 当前目录测试

当运行一些管理脚本时，可能要在根目录下运行它，特别是移动某种全局文件或进行权限改变时。一个简单的测试可以获知是否运行在根目录下。下面脚本中变量 DIRECTORY使用当前目录的命令替换操作，然后此变量值与 "/"字符串比较（/为根目录）。如果变量值与字符串不等，则用户退出脚本，退出状态为1意味错误信息产生。

```
$ pg ifpwd
#!/bin/sh
# ifpwd
DIRECTORY=`pwd`
# grab the current directory
```

China-pub.com
下载


```
if [ "$DIRECTORY" != "/" ]; then
# is it the root directory ?
# no, the direct output to standard error, which is the screen by default.
  echo "You need to be in the root directory not $DIRECTORY to run this
       script" >&2
# exit with a value of 1, an error
  exit 1
fi
```

18.3.7 文件权限测试

可以用if语句测试文件权限，下面简单测试文件 test.txt是否被设置到变量LOGNAME。

```
$ pg ifwr
#!/bin/sh
# ifwr
LOGFILE=test.txt
echo $LOGFILE
if [ ! -w "$LOGFILE" ]; then
  echo " You cannot write to $LOGFILE " >&2
fi
```

18.3.8 测试传递到脚本中的参数

if语句可用来测试传入脚本中参数的个数。使用特定变量 $#，表示调用参数的个数。可以测试所需参数个数与调用参数个数是否相等。

以下测试确保脚本有三个参数。如果没有，则返回一个可用信息到标准错误，然后代码退出并显示退出状态。如果参数数目等于 3，则显示所有参数。

```
$ pg ifparam
#!/bin/sh
# ifparam
if [ $# -lt 3 ]; then
# less than 3 parameters called, echo a usage message and exit
  echo "Usage: `basename $0`arg1 arg2 arg3" >&2
  exit 1
fi
# good, received 3 params, let's echo them
echo "arg1: $1"
echo "arg2: $2"
echo "arg3: $3"
```

如果只传入两个参数，则显示一可用信息，然后脚本退出。

```
$ ifparam cup medal
Usage:ifparam arg1 arg2 arg3
```

这次传入三个参数。

```
$ ifparam cup medal trophy
arg1: cup
arg2: medal
arg3: trophy
```

18.3.9 决定脚本是否为交互模式

有时需要知道脚本运行是交互模式（终端模式）还是非交互模式（ cron或at）。脚本也许

China-pub.com 下载

需要这个信息以决定从哪里取得输入以及输出到哪里，使用 test命令并带有-t选项很容易确认这一点。如果test返回值为1，则为交互模式。

```
$ pg ifinteractive
#!/bin/sh
# ifinteractive
if [ -t ]; then
  echo "We are interactive with a terminal"
else
  echo "We must be running from some background process probably
        cron or at "
fi
```

18.3.10 简单的if else语句

下一个if语句有可能是使用最广泛的：

```
if 条件
then
   命令1
else
   命令2
fi
```

使用if语句的else部分可在条件测试为假时采取适当动作。

18.3.11 变量设置测试

下面的例子测试环境变量EDITOR是否已设置。如果EDITOR变量为空，将此信息通知用户。如果已设置，在屏幕上显示编辑类型。

```
$ pg ifeditor
#!/bin/sh
# ifeditor
if [ -z $EDITOR ]; then
# the variable has not been set
  echo "Your EDITOR environment is not set"
else
# let's see what it is
  echo "Using $EDITOR as the default editor"
fi
```

18.3.12 检测运行脚本的用户

下面例子中，环境变量用于测试条件，即LOGNAME是否包含root值。这类语句是加在脚本开头作为一安全性准则的普遍方法。当然LOGNAME可用于测试任何有效用户。

如果变量不等root，返回信息到标准错误输出即屏幕，也就是通知用户不是 root，脚本然后退出，并带有错误值1。

如果字符串root等于LOGNAME变量，else部分后面语句开始执行。

实际上，脚本会继续进行正常的任务处理，这些语句在 fi后面，因为所有非root用户在脚本的前面测试部分已经被剔出掉了。

```
$ pg ifroot
#!/bin/sh
# ifroot
if [ "$LOGNAME" != "root" ]
# if the user is not root
then
  echo "You need to be root to run this script" >&2
  exit 1
else
# yes it is root
  echo "Yes indeed you are $LOGNAME proceed"
fi
# normal processing statements go here
```

18.3.13 将脚本参数传入系统命令

可以向脚本传递位置参数，然后测试变量。这里，如果用户在脚本名字后键入目录名，脚本将重设$1特殊变量为一更有意义的名字。即DIRECTORY。这里需测试目录是否为空，如果目录为空，ls -A将返回空，然后对此返回一信息。

```
$ pg ifdirec
#!/bin/sh
# ifdirec
# assigning $1 to DIRECTORY variable
DIRECTORY=$1
if [ "`ls -A $DIRECTORY`" = "" ] ; then
# if it's an empty string, then it's empty
  echo "$DIRECTORY is indeed empty"
else
# otherwise it is not
  echo "$DIRECTORY is not empty"
fi
```

也可以使用下面的脚本替代上面的例子并产生同样的结果。

```
$ pg ifdirec2
#!/bin/sh
# ifdirec2
DIRECTORY=$1
if [ -z "`ls -A $DIRECTORY`" ]
then
  echo "$DIRECTORY is indeed empty"
else
  echo "$DIRECTORY is not empty"
fi
```

18.3.14 null：命令用法

到目前为止，条件测试已经讲完了then和else部分，有时也许使用者并不关心条件为真或为假。

不幸的是if语句各部分不能为空——一些语句已经可以这样做。为解决此问题，shell提供了：空命令。空命令永远为真（也正是预想的那样）。回到前面的例子，如果目录为空，可以只在then部分加入命令。

China-pub.com 下载

```
$ pg ifdirectory
#!/bin/sh
# ifdirectory
DIRECTORY=$1
if [ "`ls -A $DIRECTORY`" = "" ]
then
  echo "$DIRECTORY is indeed empty"
  else :     # do nothing
fi
```

18.3.15 测试目录创建结果

现在继续讨论目录，下面的脚本接受一个参数，并用之创建目录，然后参数被传入命令行，重设给变量DIRECTORY，最后测试变量是否为空。

```
if ["$DIRECTORY"=""]
```

也可以用

```
if[$# -lt 1]
```

来进行更普遍的参数测试。

如果字符串为空，返回一可用信息，脚本退出。如果目录已经存在，脚本从头至尾走一遍，什么也没做。

创建前加入提示信息，如果键入Y或y，则创建目录，否则使用空命令表示不采取任何动作。使用最后命令状态测试创建是否成功执行，如果失败，返回相应信息。

```
$ pg ifmkdir
!/bin/sh
# ifmkdir
# parameter is passed as $1 but reassigned to DIRECTORY
DIRECTORY=$1
# is the string empty ??
if [ "$DIRECTORY" = "" ]
then
  echo "Usage :`basename $0` directory to create" >&2
  exit 1
fi
if [ -d $DIRECTORY ]
then : # do nothing
else
  echo "The directory does exist"
  echo -n "Create it now? [y..n] :"
  read ANS
  if [ "$ANS" = "y" ] || [ "$ANS" = "Y" ]
  then
    echo "creating now"
    # create directory and send all output to /dev/null
    mkdir $DIRECTORY >/dev/null 2>&1
    if [ $? != 0 ]; then
      echo "Errors creating the directory $DIRECTORY" >&2
      exit 1
    fi
  else :     # do nothing
  fi
```

China-pub.com
下载

执行上述脚本，显示：

```
$ ifmkdir dt
The directory does exist
Create it now? [y..n]:y
creating now
```

18.3.16 另一个拷贝实例

在另一个拷贝实例中，脚本传入两个参数（应该包含文件名），系统命令cp将$1拷入$2，输出至/dev/null。如果命令成功，则仍使用空命令并且不采取任何动作。

另一方面，如果失败，在脚本退出前要获知此信息。

```
$ pg ifcp2
#!/bin/sh
# ifcp2
if cp $1 $2 > /dev/null 2>&1
# successful, great do nothing
then :
else
# oh dear, show the user what files they were.
  echo "`basename $0`: ERROR failed to copy $1 to $2"
  exit 1
fi
```

脚本运行，没有拷贝错误：

```
$ cp2 myfile.lex myfile.lex.bak
```

脚本运行带有拷贝错误：

```
$ ifcp2 myfile.lexx myfile.lex.bak
ifcp2: ERROR failed to copy myfile.lexx myfile.lex.bak
```

下面的脚本用sort命令将文件accounts.qtr分类，并输出至系统垃圾堆。没人愿意观察屏幕上300行的分类页。成功之后不采取任何动作。如果失败，通知用户。

```
$ pg ifsort
#!/bin/sh
# ifsort
if sort accounts.qtr > /dev/null
# sorted. Great
then :
else
# better let the user know
  echo "`basename $0`: Oops..errors could not sort accounts.qtr"
fi
```

18.3.17 多个if语句

可能有时要嵌入if语句。为此需注意if和fi的相应匹配使用。

18.3.18 测试和设置环境变量

前面已经举例说明了如何测试环境变量EDITOR是否被设置。现在如果未设置，则进一步为其赋值，脚本如下：

China-pub.com 下载

```
$ pg ifseted
#!/bin/sh
# ifseted
# is the EDITOR set ?

if [ -z $EDITOR ] ; then
  echo "Your EDITOR environment is not set"
  echo "I will assume you want to use vi..OK"
  echo -n "Do you wish to change it now? [y..n] :"
  read ANS

  # check for an upper or lower case 'y'
  if [ "$ANS" = "y" ] || [ "$ANS" = "Y" ]; then
    echo "enter your editor type :"
    read EDITOR
    if [ -z $EDITOR ] || [ "$EDITOR" = "" ]; then
      # if EDITOR not set and no value in variable EDITOR,
      # then set it to vi
      echo "No, editor entered, using vi as default"
      EDITOR=vi
      export EDITOR
    fi

    # got a value use it for EDITOR
    EDITOR=$EDITOR
    export EDITOR
    echo "setting $EDITOR"
  fi
else
# user
  echo "Using vi as the default editor"
  EDITOR=vi
  export vi
fi
```

脚本工作方式如下：首先检查是否设置了该变量，如果已经赋值，输出信息提示使用 vi作为缺省编辑器。vi被设置为编辑器，然后脚本退出。

如果未赋值，则提示用户，询问其是否要设置该值。检验用户输入是否为大写或小写 y，输入为其他值时，脚本退出。

如果输入Y或y，再提示输入编辑类型。使用 $EDITOR=“ ”测试用户是否未赋值和未点击return键。一种更有效的方法是使用 -z $EDITOR方法，本文应用了这两种方法。如果测试失败，返回信息到屏幕，即使用 vi做缺省编辑器，因而EDITOR赋值为vi。

如果用户输入了一个名字到变量 EDITOR，则使用它作为编辑器并马上让其起作用，即导出变量EDITOR。

18.3.19 检测最后命令状态

前面将目录名传入脚本创建了一个目录，脚本然后提示用户是否应创建目录。下面的例子创建一个目录，并从当前目录将所有 *.txt文件拷入新目录。但是这段脚本中用最后状态命令检测了每一个脚本是否成功执行。如果命令失败则通知用户。

```
$ pg ifmkdir2
#!/bin/sh
# ifmkdir2
DIR_NAME=testdirec
# where are we ?
THERE=`pwd`
# send all output to the system dustbin
mkdir $DIR_NAME > /dev/null 2>&1
# is it a directory ?
if [ -d $DIR_NAME ]; then
# can we cd to the directory
  cd $DIR_NAME
  if [ $? = 0 ]; then
  # yes we can
    HERE=`pwd`
    cp $THERE/*.txt $HERE
  else
    echo "Cannot cd to $DIR_NAME"  >&2
    exit 1
  fi
else
  echo "$cannot create directory $DIR_NAME" >&2
  exit 1
fi
```

18.3.20 增加和检测整数值

下面的例子进行数值测试。脚本包含了一个计数集，用户将其赋予一个新值就可改变它。脚本然后将当前值100加入一个新值。工作流程如下：

用户输入一个新值改变其值，如果键入回车键，则不改变它，打印当前值，脚本退出。

如果用户用y或Y响应新值，将提示用户输入增量。如果键入回车键，原值仍未变。键入一个增量，首先测试是否为数字，如果是，加入计数 COUNTOR中，最后显示新值。

```
$ pg ifcounter
#!/bin/sh
# ifcounter
COUNTER=100
echo "Do you wish to change the counter value currently set at $COUNTER ?
      [y..n] :"
read ANS
if [ "$ANS" = "y" ] || [ "$ANS" = "Y" ]; then
# yes user wants to change the value
  echo "Enter a sensible value "
  read VALUE
  # simple test to see if it's numeric, add any number to VALUE,
  # then check out return
  # code
  expr $VALUE + 10 > /dev/null 2>&1
    STATUS=$?
  # check return code of the expr
  if [ "$VALUE" = "" ] || [ "$STATUS" != "0" ]; then
  # send errors to standard error
    echo " You either entered nothing or a non-numeric " >&2
    echo " Sorry now exiting..counter stays at $COUNTER" >&2
```

China-pub.com 下载

```
    exit 1
  fi
# if we are here, then it's a number, so add it to COUNTER
  COUNTER=`expr $COUNTER + $VALUE`
  echo " Counter now set to $COUNTER"
else
# if we are here then user just hit return instead of entering a number
# or answered n to the change a value prompt
  echo " Counter stays at $COUNTER"
fi
```

运行结果如下：

```
$ ifcount
Do you wish to change the counter value currently set at 100 ? [y..n]:n
Counter stays at 100


$ ifcount
Do you wish to change the counter value currently set at 100 ? [y..n]:y
Enter a sensible value
fdg
 You either entered nothing or a non-numeric
 Sorry now exiting..counter stays at 100


$ ifcount
Do you wish to change the counter value currently set at 100 ? [y..n]:y
Enter a sensible value  250
Counter now set to 350
```

18.3.21 简单的安全登录脚本

以下是用户登录时启动应用前加入相应安全限制功能的基本框架。首先提示输入用户名和密码，如果用户名和密码均匹配脚本中相应字符串，用户登录成功，否则用户退出。

脚本首先设置变量为假——总是假定用户输入错误， stty当前设置被保存，以便隐藏passwd域中字符，然后重新保存stty设置。

如果用户ID和密码正确（密码是mayday），明亮INVALID_USER和INVALID_PASSWD设置为no表示有效用户或密码，然后执行测试，如果两个变量其中之一为 yes，缺省情况下，脚本退出用户。

键入有效的ID和密码，用户将允许进入。这是一种登录脚本的基本框架。下面的例子中有效用户ID为dave或pauline。

```
$ pg ifpass
#!/bin/sh
# ifpass
# set the variables to false
INVALID_USER=yes
INVALID_PASSWD=yes
# save the current stty settings
SAVEDSTTY=`stty -g`
echo "You are logging into a sensitive area"
echo -n "Enter your ID name :"
read NAME
# hide the characters typed in
stty -echo
```

```
echo "Enter your password :"
read PASSWORD
# back on again
stty $SAVEDSTTY
if [ "$NAME" = "dave" ] || [ "$NAME" = "pauline" ]; then
  # if a valid then set variable
  INVALID_USER=no
fi

if [ "$PASSWORD" = "mayday" ]; then
# if valid password then set variable
INVALID_PASSWD=no
fi
if [ "$INVALID_USER" = "yes" -o "$INVALID_PASSWD" = "yes" ]; then
  echo "`basename $0 :` Sorry wrong password or userid"
  exit 1
fi
# if we get here then their ID and password are OK.
echo "correct user id and password given"
```

如果运行上述脚本并给一个无效用户：

```
$ ifpass
You are logging into a sensitive area
Enter your ID name  : dave
Enter your password :
ifpass :Sorry wrong password or userid
```

现在给出正确的用户和密码：

```
$ ifpass
You are logging into a sensitive area
Enter your ID name  : dave
Enter your password :
correct user id and password given
```

18.3.22 elif用法

if then else语句的elif部分用于测试两个以上的条件。

18.3.23 使用elif进行多条件检测

使用一个简单的例子，测试输入脚本的用户名。脚本首先测试是否输入一个名字，如果没有，则什么也不做。如果输入了，则用 elif测试是否匹配root、louise或dave，如果不匹配其中任何一个，则打印该名字，通知用户不是 root、louise或dave。

```
$ pg ifelif
#!/bin/sh
# ifelif
echo -n "enter your login name :"
read NAME
# no name entered do not carry on
if [ -z $NAME ] || [ "$NAME" = "" ]; then
  echo "You did not enter a name"
elif
  # is the name root
  [ "$NAME" = "root" ]; then
```

China-pub.com 下载

```
    echo "Hello root"
  elif
    # or is it louise
    [ $NAME = "louise" ]; then
    echo "Hello louise"
  elif
    # or is it dave
    [ "$NAME" = "dave" ]; then
    echo "Hello dave"
  else
    # no it's somebody else
    echo "You are not root or louise or dave but hi $NAME"
  fi
```

运行上述脚本，给出不同信息，得结果如下：

```
$ ifelif
enter your login name : dave
Hello dave

$ ifelif
enter your login name :
You did not enter a name

$ ifelif2
enter your login name : peter
You are not root or louise or dave but hi peter
```

18.3.24 多文件位置检测

假定要定位一个用户登录文件，已知此文件在 /usr/opts/audit/logs或/usr/local/audit/logs中，具体由其安装人决定。在定位此文件前，首先确保文件可读，此即脚本测试部分。如果未找到文件或文件不可读，则返回错误信息。脚本如下：

```
$ pg ifcataudit
#!/bin/sh
# ifcataudit
# locations of the log file
LOCAT_1=/usr/opts/audit/logs/audit.log
LOCAT_2=/usr/local/audit/audit.logs

if [ -r $LOCAT_1 ]; then
  # if it is in this directory and is readable then cat it
  echo "Using LOCAT_1"
  cat $LOCAT_1
elif
  # else it then must be in this directory, and is it readable
  [ -r $LOCAT_2 ]
then
  echo "Using LOCAT_2"
  cat $LOCAT_2
else
  # not in any of the directories...
  echo "`basename $0`: Sorry the audit file is not readable or cannot be
       located." >&2
  exit 1
```

```
fi
```

运行上面脚本，如果文件在上述两个目录之一中并且可读，将可以找到它。如果不是，返回错误并退出，下面结果失败，因为假想的文件并不存在。

```
$ ifcataudit
ifcataudit: Sorry the audit file is not readable or cannot be located.
```

18.4 case语句

case语句为多选择语句。可以用case语句匹配一个值与一个模式，如果匹配成功，执行相匹配的命令。case语句格式如下：

```
case  值  in
模式1}
   命令1
   ...
   ;;
模式2）
   命令2
   ...
   ；；
esac
```

case工作方式如上所示。取值后面必须为单词in，每一模式必须以右括号结束。取值可以为变量或常数。匹配发现取值符合某一模式后，其间所有命令开始执行直至；；。

取值将检测匹配的每一个模式。一旦模式匹配，则执行完匹配模式相应命令后不再继续其他模式。如果无一匹配模式，使用星号*捕获该值，再接受其他输入。

模式部分可能包括元字符，与在命令行文件扩展名例子中使用过的匹配模式类型相同，即：

* 任意字符。

？ 任意单字符。

[..] 类或范围中任意字符。

下面举例说明。

18.4.1 简单的case语句

下面的脚本提示输入1到5，输入数字传入case语句，变量ANS设置为case取值测试变量ANS，ANS将与每一模式进行比较。

如果匹配成功，则执行模式里面的命令直至；;，这里只反馈非用户数字选择的信息，然后case退出，因为匹配已找到。

进程在case语句后仍可继续执行。

如果匹配未找到，则使用*模式捕获此情况，这里执行错误信息输出。

```
$ pg caseselect
#!/bin/sh
# caseselect
```

China-pub.com 下载

```
echo -n "enter a number from 1 to 5  :"
read ANS
  case $ANS in
  1) echo "you select 1"
    ;;
  2) echo "you select 2"
    ;;
  3) echo "you select 3"
    ;;
  4) echo "you select 4"
    ;;
  5) echo "you select 5"
    ;;
  *) echo "`basename $0`: This is not between 1 and 5" >&2
    exit 1
    ;;
  esac
```

给出不同输入，运行此脚本。

```
$ caseselect
enter a number from 1 to 5 : 4
you select 4
```

使用模式*捕获范围之外的取值情况。

```
$ caseselect
enter a number from 1 to 5  :pen
caseselect: This is not between 1 and 5
```

18.4.2 对匹配模式使用|

使用case时，也可以指定“|”符号作为或命令，例如vt100|vt102匹配模式vt100或vt102。下面的例子中，要求用户输入终端类型。如果输入为vt100或vt102，将匹配模式‘vt100|vt102’，执行命令是设置TERM变量为vt100。如果用户输入与模式不匹配，*用来捕获输入，其中命令为将TERM设置为vt100。最后在case语句外，导出TERM变量。由于使用*模式匹配，无论用户输入什么，TERM都将有一个有效的终端类型值。

```
$ pg caseterm
#!/bin/sh
# caseterm
echo " choices are.. vt100, vt102, vt220"
echo -n "enter your terminal type :"
read TERMINAL
  case $TERMINAL in
  vt100|vt102) TERM=vt100
    ;;
  vt220) TERM=vt220
    ;;
  *) echo "`basename $0` : Unknown response" >&2
    echo "setting it to vt100 anyway, so there"
    TERM=vt100
    ;;
  esac
  export TERM
echo "Your terminal is set to $TERM"
```

运行脚本，输入一无效终端类型，

```
$ caseterm
 choices are.. vt100, vt102, vt220
enter your terminal type :vt900
caseterm : Unknown response
setting it to vt100 anyway, so there
Your terminal is set to vt100
```

如果输入一正确的终端类型，

```
$ case2
 choices are.. vt100, vt102, vt220
enter your terminal type :vt220
Your terminal is set to vt220
```

无论怎样，一个有效的终端类型被赋予用户。

18.4.3 提示键入y或n

case的一个有效用法是提示用户响应以决定是否继续进程。这里提示输入 y以继续处理， n退出。如果用户输入Y、y或yes，处理继续执行case语句后面部分。如果用户输入N、n或no或其他响应，用户退出脚本。

```
$ pg caseans
#!/bin/sh
# caseans
echo -n "Do you wish to proceed [y..n] :"
read ANS
  case $ANS in
  y|Y|yes|Yes) echo "yes is selected"
    ;;
  n|N) echo "no is selected"
    exit 0    # no error so only use exit 0 to terminate
    ;;
  *) echo "`basename $0` : Unknown response" >&2
    exit 1
    ;;
  esac
# if we are here then a y|Y|yes|Yes was selected only.
```

运行脚本，输入无效响应，得结果：

```
$ caseans
Do you wish to proceed [y..n] :df
caseans : Unknown response
```

给出有效响应：

```
$ caseans
Do you wish to proceed [y..n] :y
yes is selected
```

18.4.4 case与命令参数传递

可以使用case控制到脚本的参数传递。

下面脚本中，测试特定变量 $#，它包含传递的参数个数，如果不等于 1，退出并显示可用信息。

China-pub.com 下载

然后case语句捕获下列参数：passwd、start、stop或help，相对于每一种匹配模式执行进一步处理脚本。如果均不匹配，显示可用信息到标准错误输出。

```
$ pg caseparam
#!/bin/sh
# caseparam
if [ $# != 1 ]; then
  echo "Usage:`basename $0`[start|stop|help]"  >&2
  exit 1
fi
# assign the parameter to the variable OPT
OPT=$1
  case $OPT in
  start) echo "starting..`basename $0`"
    # code here to start a process
    ;;
  stop) echo "stopping..`basename $0`"
    # code here to stop a process
    ;;
  help)
    # code here to display a help page
    ;;
  *) echo "Usage:`basename $0`[start|stop|help]"
    ;;
  esac
```

运行脚本，输入无效参数。

```
$ caseparam what
Usage:caseparam [start|stop|help]
```

输入有效参数，结果为：

```
$ caseparam stop
stopping..caseparam
```

18.4.5 捕获输入并执行空命令

不一定要在匹配模式后加入命令，如果你原本不想做什么，只是在进一步处理前过滤出意外响应，这样做是一种好办法。

如果要运行对应于一个会计部门的帐目报表，必须首先在决定运行报表的类型前确认用户输入一个有效的部门号，匹配所有可能值，其他值无效。用case可以很容易实现上述功能。

下面的脚本中如果用户输入部门号不是234、453、655或454，用户退出并返回可用信息。一旦响应了用户的有效部门号，脚本应用同样的技术取得报表类型，在 case语句末尾显示有效的部门号和报表类型。脚本如下：

```
$ pg casevalid
!/bin/sh
# casevalid
TYPE=""
echo -n "enter the account dept No: :"
read ACC
  case $ACC in
    234);;
    453);;
```

China-pub.com
下载


```
      655);;
      454);;
    *) echo "`basename $0`: Unknown dept No:"  >&2
      echo "try..234,453,655,454"
      exit 1
      ;;
    esac

  # if we are here, then we have validated the dept no
  echo " 1 . post"
  echo " 2 . prior"
  echo -n "enter the type of report: "
  read ACC_TYPE
    case $ACC_TYPE in
    1)TYPE=post;;
    2)TYPE=prior;;
    *)  echo "`basename $0`: Unknown account type." >&2
      exit 1
      ;;
    esac

  # if we get here then we are validated!
  echo "now running report for dept $ACC for the type $TYPE"
  # run the command report..
```

输入有效部门号：

```
$ casevalid
enter the account dept No: :234
 1 . post
 2 . prior
enter the type of report:2
now running report for dept 234 for the type  prior
```

输入无效部门号：

```
$ casevalid
enter the account dept No: :432
casevalid: Unknown dept No:
try..234,453,655,454
```

输入无效的报表类型：

```
$ casevalid
enter the account dept No: :655
 1 . post
 2 . prior
enter the type of report:4
casevalid: Unknown account type.
```

18.4.6 缺省变量值

如果在读变量时输入回车键，不一定总是退出脚本。可以先测试是否已设置了变量，如果未设置，可以设置该值。

下面的脚本中，要求用户输入运行报表日期。如果用户输入回车键，则使用缺省日期星期六，并设置为变量when的取值。

如果用户输入另外一天，这一天对于case语句是运行的有效日期，即星期六、星期四、星

China-pub.com 下载

期一。注意，这里结合使用了日期缩写作为捕获的可能有效日期。

脚本如下：

```
$ pg caserep
#!/bin/sh
# caserep
echo "      Weekly Report"
echo -n "What day do you want to run report [Saturday] :"

# if just a return is hit then except default which is Saturday
read WHEN
echo "validating..${WHEN:="Saturday"}"
  case $WHEN in
  Monday|MONDAY|mon)
    ;;
  Sunday|SUNDAY|sun)
    ;;
  Saturday|SATURDAY|sat)
    ;;
  *) echo " Are you nuts!, this report can only be run on " >&2
    echo " on a Saturday, Sunday or Monday" >&2
    exit 1
    ;;
  esac
echo "Report to run on $WHEN"
# command here to submitted actual report run
```

对于正确输入：

```
$ caserep
      Weekly Report
What day do you want to run report [Saturday] :
validating..Saturday
Report to run on Saturday
```

对于错误输入：

```
$ caserep
      Weekly Report
What day do you want to run report [Saturday] :Tuesday
validating..Tuesday
 Are you nuts!, this report can only be run on
 on a Saturday, Sunday or Monday
```

可以推断出case语句有时与if then else语句功能相同，在某些条件下，这种假定是正确的。

18.5 for循环

for循环一般格式为：

for 变量名in列表

do

命令1

命令2…

done

China-pub.com 下载

当变量值在列表里，for循环即执行一次所有命令，使用变量名访问列表中取值。命令可为任何有效的shell命令和语句。变量名为任何单词。In列表用法是可选的，如果不用它，for循环使用命令行的位置参数。

in列表可以包含替换、字符串和文件名，下面看一些例子。

18.5.1 简单的for循环

此例仅显示列表1 2 3 4 5，用变量名访问列表。

```
$ pg for_i
#!/bin/sh
# for_i
for loop in 1 2 3 4 5
do
  echo $loop
done
```

运行上述脚本，输出：

```
$ for_i
1
2
3
4
5
```

18.5.2 打印字符串列表

下面for循环中，列表包含字符串“ orange red blue grey ”，命令为echo，变量名为loop，echo命令使用$loop反馈出列表中所有取值，直至列表为空。

```
$ pg forlist
#!/bin/sh
# forlist
for loop in "orange red blue grey"
do
  echo $loop
done

$ forlist
orange red blue grey
```

也可以在循环体中结合使用变量名和字符串。

```
echo " this is the fruit $loop"
```

结果为：

```
This is the fruit orange red blue grey
```

18.5.3 对for循环使用ls命令

这个循环执行ls命令，打印当前目录下所有文件。

```
$ pg forls
#!/bin/sh
```

China-pub.com 下载

```
# forls
for loop in `ls `
do
  echo $loop
done

$ forls
array
arrows
center
center1
center2
centerb
```

18.5.4 对for循环使用参数

在for循环中省去in列表选项时，它将接受命令行位置参数作为参数。实际上即指明：

```
for  params  in"$@"
```

或

```
for  params  in"$*"
```

下面的例子不使用in列表选项，for循环查看特定参数$@或$*，以从命令行中取得参数。

```
$ pg forparam2
#!/bin/sh
# forparam2
for params
do
  echo "You supplied $params as a command line option"
done
  echo $params
done

$ forparam2 myfile1 myfile2 myfile3
 You supplied myfile1 as a command line option
 You supplied myfile2 as a command line option
 You supplied myfile3 as a command line option
```

下面的脚本包含in"$@"，结果与上面的脚本相同。

```
$ pg forparam3
#!/bin/sh
# forparam3
for params in "$@"
do
  echo "You supplied $params as a command line option"
done
  echo $params
done
```

对上述脚本采取进一步动作。如果要查看一系列文件，可在 for循环里使用find命令，利用命令行参数，传递所有要查阅的文件。

```
$ pg forfind
#!/bin/sh
# forfind
```

China-pub.com
下载


```
for loop
do
  find / -name $loop -print
done
```

脚本执行时，从命令行参数中取值并使用 find命令，这些取值形成-name选项的参数值。

```
$ forfind passwd LPSO.AKSOP
/etc/passwd
/etc/pam.d/passwd
/etc/uucp/passwd
/usr/bin/passwd
/usr/local/accounts/LPSO.AKSOP
```

18.5.5 使用for循环连接服务器

因为for循环可以处理列表中的取值，现设变量为网络服务器名称，并使用 for循环连接每一服务器。

```
$ pg forping
#!/bin/sh
# forping
HOSTS="itserv dnssevr acctsmain ladpd ladware"
for loop in $HOSTS
do
  ping -c 2 $loop
done
```

18.5.6 使用for循环备份文件

可以用 for循环备份所有文件，只需将变量作为 cp命令的目标参数。这里有一变量 .bak，当在循环中使用cp命令时，它作为此命令目标文件名。列表命令为 ls。

18.5.7 多文件转换

匹配所有以LPSO开头文件并将其转换为大写。这里使用了 ls和cat命令。ls用于查询出相关文件，cat用于将之管道输出至tr命令。目标文件扩展名为.UC，注意在for循环中使用ls命令时反引号的用法。

```
$ pg forUC
#!/bin/sh
# forUC
for files in `ls LPSO*`
do
  cat $files |tr "[a-z]" "[A-Z]" >$files.UC
done
```

18.5.8 多sed删除操作

下面的例子中， sed用于删除所有空文件，并将输出导至以 .HOLD.mv为扩展名的新文件中，mv将这些文件移至初始文件中。

```
#!/bin/sh
# forsed
for files in `ls LPSO*`
do
  sed -e "/^$/d" $files >$files.HOLD
  mv $files.HOLD $files
done
```

18.5.9 循环计数

前面讨论expr时指出，循环时如果要加入计数，使用此命令。下面使用 ls在for循环中列出文件及其数目。

```
$ pg forcount
#!/bin/sh
# forcount
counter=0
for files in *
do
  # increment
  counter=`expr $counter + 1`
done
  echo "There are $counter files in `pwd` we need to process"

$ forcount
There are 45 files in /apps/local we need to process
```

使用wc命令可得相同结果。

```
$ ls |wc -l
      45
```

18.5.10 for循环和本地文档

在for循环体中可使用任意命令。下面的例子中，一个变量包含所有当前登录用户。使用who命令并结合awk语言可实现此功能。然后 for循环循环每一用户，给其发送一个邮件，邮件信息部分用一个本地文档完成。

```
$ pg formailit
#!/bin/sh
```

China-pub.com
下载


```
# formailit
WHOS_ON=`who -u | awk '{print $1}'`
for user in $WHOS_ON
do
mail $user << MAYDAY
Dear Colleagues,
It's my birthday today, see you down the
club at 17:30 for a drink.

See ya.
$LOGNAME
MAYDAY
Done
```

上述脚本的邮件信息输出为：

```
$ pg mbox
Dear Colleagues,
It's my birthday today, see you down the
club at 17:30 for a drink.

See ya.
dave
```

18.5.11 for循环嵌入

嵌入循环可以将一个for循环嵌在另一个for循环内：

```
for  变量名1  in列表1
do
    for  变量名2  in  列表2
    do
          命令1
          ...
    done
done
```

下面脚本即为嵌入for循环，这里有两个列表APPS和SCRIPTS。第一个包含服务器上应用的路径，第二个为运行在每个应用上的管理脚本。对列表 APPS上的每一个应用，列表SCRIPTS里的脚本将被运行，脚本实际上为后台运行。脚本使用 tee命令在登录文件上放一条目，因此输出到屏幕的同时也输出到一个文件。查看输出结果就可以看出嵌入 for循环怎样使用列表SCRIPTS以执行列表APPS上的处理。

```
$ pg audit_run
#!/bin/sh
# audit_run
APPS="/apps/accts /apps/claims /apps/stock /apps/serv"
SCRIPTS="audit.check report.run cleanup"
LOGFILE=audit.log
MY_DATE=`date +%H:%M" on "%d/%m%Y`

# outer loop
for loop in $APPS
do
```

China-pub.com 下载

```
  # inner loop
  for loop2 in $SCRIPTS
  do
    echo "system $loop now running $loop2 at $MY_DATE" | tee -a $LOGFILE
    $loop $loop2 &

  done
done
```

```
$ audit_run
system /apps/accts now running audit.check at 20:33 on 23/051999
system /apps/accts now running report.run at 20:33 on 23/051999
system /apps/accts now running cleanup at 20:33 on 23/051999
system /apps/claims now running audit.check at 20:33 on 23/051999
system /apps/claims now running report.run at 20:33 on 23/051999
system /apps/claims now running cleanup at 20:34 on 23/051999
system /apps/stock now running audit.check at 20:34 on 23/051999
system /apps/stock now running report.run at 20:34 on 23/051999
system /apps/stock now running cleanup at 20:34 on 23/051999
system /apps/serv now running audit.check at 20:34 on 23/051999
system /apps/serv now running report.run at 20:34 on 23/051999
system /apps/serv now running cleanup at 20:34 on 23/051999
```

18.6 until循环

until循环执行一系列命令直至条件为真时停止。 until循环与while循环在处理方式上刚好相反。一般while循环优于until循环，但在某些时候—也只是极少数情况下， until循环更加有用。

until循环格式为：

```
until  条件
          命令1
          ...
done
```

条件可为任意测试条件，测试发生在循环末尾，因此循环至少执行一次—请注意这一点。

下面是一些实例。

18.6.1 简单的until循环

这段脚本不断的搜寻who命令中用户root，变量IS-ROOT保存grep命令结果。

如果找到了root，循环结束，并向用户 simon发送邮件，通知他用户 root已经登录，注意这里sleep命令用法，它经常用于 until循环中，因为必须让循环体内命令睡眠几秒钟再执行，否则会消耗大量系统资源。

```
$ pg until_who
#!/bin/sh
# until_who
IS_ROOT=`who | grep root`
until [ "$IS_ROOT" ]
do
```

```
  sleep 5
done
echo "Watch it. roots in " | mail simon
```

18.6.2 监视文件

下面例子中，until循环不断挂起做睡眠，直至文件 /tmp/monitor.lck被删除。文件删除后，脚本进入正常处理过程。

```
$ pg until_lck
!/bin/sh
# until_lck
LOCK_FILE=/tmp/process.LCK
until  [ !  -f $LOCK_FILE ]

do
  sleep 1
done
echo "file deleted "
# normal processing now, file is present
```

上述例子是使脚本与其他处理过程协调工作的一种方法。还有另外一种方法使脚本间互相通信。假定有另一段脚本 process.main用于搜集本地网络所有机器的信息并将之放入一个报表文件。

当脚本 process.main运行时，创建了一个 LCK文件（锁文件），上面脚本必须接收process.main搜集的信息，但是如果 process仍然在修改报表文件时试图处理该文件就不太好了。

为克服这些问题，脚本process.main创建了一个LCK文件，当它完成时，就删除此文件。

上述脚本将挂起，等待 LCK文件被删除，一旦 LCK文件删除，上述脚本即可处理报表文件。

18.6.3 监视磁盘空间

until循环做监视条件也很有用。假定要监视文件系统容量，当它达到一定水平时通知超级用户。

下面的脚本监视文件系统 /logs，不断从变量 $LOOK_OUT中抽取信息，$LOOK_OUT包含使用awk和grep得到的/logs容量。

如果容量达到90%，触发命令部分，向超级用户发送邮件，脚本退出。必须退出，如果不退出，条件保持为真（例如，容量总是保持在 90%以上），将会不断的向超级用户发送邮件。

```
$ pg until_mon
#!/bin/sh
# until_mon
# get percent column and strip off header row from df
LOOK_OUT=`df |grep /logs | awk '{print $5}'| sed 's/%//g'`
echo $LOOK_OUT
until [ "$LOOK_OUT" -gt "90" ]
do
```

China-pub.com 下载

```
  echo "Filesystem..logs is nearly full" | mail root
  exit 0
done
```

18.7 while循环

while循环用于不断执行一系列命令，也用于从输入文件中读取数据，其格式为：

```
while 命令
do
    命令1
    命令2
      ...
done
```

虽然通常只使用一个命令，但在 while和do之间可以放几个命令。命令通常用作测试条件。

只有当命令的退出状态为 0时，do和done之间命令才被执行，如果退出状态不是 0，则循环终止。

命令执行完毕，控制返回循环顶部，从头开始直至测试条件为假。

18.7.1 简单的while循环

以下是一个基本的 while循环，测试条件是：如果 COUNTER小于5，那么条件返回真。COUNTER从0开始，每次循环处理时， COUNTER加1。

```
$ pg whilecount
#!/bin/sh
# whilecount
COUNTER=0
# does the counter = 5 ?
while [ $COUNTER -lt 5 ]
do
  # add one to the counter
  COUNTER=`expr $COUNTER + 1`
  echo $COUNTER
done
```

运行上述脚本，返回数字1到5，然后终止。

```
$ whilecount
1
2
3
4
5
```

18.7.2 使用while循环读键盘输入

while循环可用于读取键盘信息。下面的例子中，输入信息被设置为变量 FILM，按<Ctrl-D>结束循环。

```
$ pg whileread
```

```
#!/bin/sh
# whileread
echo " type <CTRL-D> to terminate"
echo -n "enter your most liked film :"
while read FILM
do
  echo "Yeah, great film the $FILM"
done
```

脚本运行时，输入可能是：

```
$ whileread
enter your most liked film: Sound of Music
Yeah, great film the Sound of Music
<CTRL-D>
```

18.7.3 用while循环从文件中读取数据

while循环最常用于从一个文件中读取数据，因此编写脚本可以处理这样的信息。

假定要从下面包含雇员名字、从属部门及其ID号的一个文件中读取信息。

```
$ pg names.txt
Louise Conrad:Accounts:ACC8987
Peter James:Payroll:PR489
Fred Terms:Customer:CUS012
James Lenod:Accounts:ACC887
Frank Pavely:Payroll:PR489
```

可以用一个变量保存每行数据，当不再有读取数据时条件为真。 while循环使用输入重定向以保证从文件中读取数据。注意整行数据被设置为单变量 $LINE.。

```
$ pg whileread
#!/bin/sh
# whileread
while read LINE
do
  echo $LINE
done < names.txt

$ whileread
Louise Conrad:Accounts:ACC8987
Peter James:Payroll:PR489
Fred Terms:Customer:CUS012
James Lenod:Accounts:ACC887
Frank Pavely:Payroll:PR489
```

18.7.4 使用IFS读文件

输出时要去除冒号域分隔符，可使用变量 IFS。在改变它之前保存 IFS的当前设置。然后在脚本执行完后恢复此设置。使用 IFS可以将域分隔符改为冒号而不是空格或 tab键。这里有3个域需要加域分隔，即NAME、DEPT和ID。

为使输出看起来更清晰，对echo命令使用tab键将域分隔得更开一些，脚本如下：

```
$ pg   whilereadifs
#!/bin/sh
# whilereadifs
# save the setting of IFS
```

China-pub.com 下载

```
SAVEDIFS=$IFS
# assign new separator to IFS
IFS=:
while read NAME DEPT ID
do
  echo -e "$NAME\t $DEPT\t $ID"
done < names.txt
# restore the settings of IFS
IFS=$SAVEDIFS
```

脚本运行，输出果然清晰多了。

```
$ whilereadifs
Louise Conrad        Accounts    ACC8987
Peter James          Payroll     PR489
Fred Terms           Customer    CUS012
James Lenod          Accounts    ACC887
Frank Pavely         Payroll     PR489
```

18.7.5 带有测试条件的文件处理

大部分while循环里都带有一些测试语句，以决定下一步的动作。

这里从人员文件中读取数据，打印所有细节到一个保留文件中，直至发现 James Lenod，脚本退出。测试前反馈的信息要确保“ James Lenod ”加入保留文件中。

注意，所有变量在脚本顶端被设置完毕。这样当不得不对变量进行改动时可以节省时间和输入。所有编辑都放在脚本顶端，而不是混于整个脚本间。

```
$ pg whileread_file
#!/bin/sh
# whileread_file
# initialise variables
SAVEDIFS=$IFS
IFS=:
HOLD_FILE=hold_file
NAME_MATCH="James Lenod"
INPUT_FILE=names.txt

# create a new HOLD_FILE each time, in case script is continuously run
>$HOLD_FILE
while read NAME DEPT ID
do
  # echo all information into holdfile with redirection
  echo $NAME $DEPT $ID >>$HOLD_FILE
  # is it a match ???
  if [ "$NAME" = "$NAME_MATCH" ]; then
    # yes then nice exit
    echo "all entries up to and including $NAME_MATCH are in $HOLD_FILE"
    exit 0
  fi
done < $INPUT_FILE
# restore IFS
IFS=$SAVEDIFS
```

还可以采取进一步动作，列出多少个雇员属于同一部门。这里保持同样的读方式。假定每个域都有一个变量名，然后在case语句里用expr增加每行匹配脚本。任何发现的未知部门知

识反馈到标准错误中，如果一个无效部门出现，没有必要退出。

```
$ pg whileread_cond
!/bin/sh
# whileread_cond
# initialise variables
ACC_LOOP=0; CUS_LOOP=0; PAY_LOOP=0;

SAVEDIFS=$IFS
IFS=:
while read NAME DEPT ID
do
  # increment counter for each matched dept.
  case $DEPT in
  Accounts) ACC_LOOP=`expr $ACC_LOOP + 1`
    ACC="Accounts"
    ;;
  Customer) CUS_LOOP=`expr $CUS_LOOP + 1`
    CUS="Customer"
    ;;
  Payroll) PAY_LOOP=`expr $PAY_LOOP + 1`
    PAY="Payroll"
    ;;
  *) echo "`basename $0`: Unknown department $DEPT" >&2
    ;;
  esac
done < names.txt
IFS=$SAVEDIFS
echo "there are $ACC_LOOP employees assigned to $ACC dept"
echo "there are $CUS_LOOP employees assigned to $CUS dept"
echo "there are $PAY_LOOP employees assigned to $PAY dept"
```

运行脚本，输出：

```
$ whileread_cond
there are 2 employees assigned to Accounts dept
there are 1 employees assigned to Customer dept
there are 2 employees assigned to Payroll dept
```

18.7.6 扫描文件行来进行数目统计

一个常用的任务是读一个文件，统计包含某些数值列的数值总和。下面的文件包含有部门STAT和GIFT所卖的商品数量。

```
$ pg total.txt
STAT    3444
GIFT     233
GIFT     252
GIFT     932
STAT     212
STAT     923
GIFT     129
```

现在的任务是要统计部门 GIFT所卖的各种商品数量。使用 expr保存统计和，看下面的expr语句。变量LOOP和TOTAL首先在循环外初始化为0，循环开始后， ITEMS加入TOTAL，第一次循环只包含第一种商品，但随着过程继续， ITEMS逐渐加入TOTAL。

China-pub.com 下载

下面的expr语句不断增加计数。

```
LOOP=0
TOTAL=0
...
while...
TOTAL=`expr $TOTAL + $ITEMS`
ITEMS=`expr $ITEMS + 1`
done
```

使用expr语句时容易犯的一个错误是开始忘记初始化变量。

```
LOOP=0
TOTAL=0
```

如果真的忘了初始化，屏幕上将布满 expr错误。

如果愿意，可以在循环内初始化循环变量。

```
TOTAL=`expr ${TOTAL:=0} + ${ITEMS}`
```

上面一行如果变量TOTAL未赋值，将其初始化为0。这是在expr里初始化变量的第一个例子。另外在循环外要打印出最后总数。

```
$ pg $total
#!/bin/sh
# total
# init variables
LOOP=0
TOTAL=0
COUNT=0

echo "Items  Dept"
echo "____________"
while read DEPT ITEMS
do
  # keep a count on total records read
  COUNT=`expr $COUNT + 1`
  if [ "$DEPT" = "GIFT" ]; then
    # keep a running total
    TOTAL=`expr $TOTAL + $ITEMS`
    ITEMS=`expr $ITEMS + 1`
    echo -e "$ITEMS\t$DEPT"
  fi
  #echo $DEPT $ITEMS
done < total.txt
echo "============"
echo $TOTAL
echo "There were $COUNT entries altogether in the file"
```

运行脚本，得到：

```
$ total
Items  Dept
____________
234      GIFT
253      GIFT
933      GIFT
130      GIFT
============
```

China-pub.com 下载


```
1546
There were 7 entries altogether in the file
```

18.7.7 每次读一对记录

有时可能希望每次处理两个记录，也许可从记录中进行不同域的比较。每次读两个记录很容易，就是要在第一个while语句之后将第二个读语句放在其后。使用这项技术时，不要忘了不断进行检查，因为它实际上读了大量的记录。

```
$ pg record.txt
record 1
record 2
record 3
record 4
record 5
record 6
```

每次读两个记录，下面的例子对记录并不做实际测试。

脚本如下：

```
$ pg readpair
#!/bin/sh
# readpair
# first record
while read rec1
do
  # second record
  read rec2
  # further processing/testing goes here to test or compare both records
  echo "This is record one of a pair :$rec1"
  echo "This is record two of a pair :$rec2"
  echo "----------------------------"
done < record.txt
```

首先来检查确实读了很多记录，可以使用wc命令：

```
$ cat record.txt | wc -l
      6
```

共有6个记录，观察其输出：

```
$ readpair
This is record one of a pair :record 1
This is record two of a pair :record 2
-----------------------------
This is record one of a pair :record 3
This is record two of a pair :record 4
-----------------------------
This is record one of a pair :record 5
This is record two of a pair :record 6
-----------------------------
```

18.7.8 忽略#字符

读文本文件时，可能要忽略或丢弃遇到的注释行，下面是一个典型的例子。

假定要使用一般的while循环读一个配置文件，可拣选每一行，大部分都是实际操作语句。有时必须忽略以一定字符开头的行，这时需要用 case语句，因为#是一个特殊字符，最好首先

China-pub.com 下载

用反斜线屏蔽其特殊意义，在#符号后放一个星号*，指定*后可包含任意字符。

配置文件如下：

```
$ pg config
# THIS IS THE SUB SYSTEM AUDIT CONFIG FILE
# DO NOT EDIT!!!!.IT WORKS
#
# type of admin access
AUDITSCM=full
# launch place of sub-systems
AUDITSUB=/usr/opt/audit/sub
# serial hash number of product
HASHSER=12890AB3
# END OF CONFIG FILE!!!
```

忽略#符号的实现脚本如下：

```
$ pg ignore_hash
#!/bin/sh
# ignore_hash
INPUT_FILE=config
if [-s $INPUT_FILE ]; then
  while read LINE
  do
    case $LINE in
    \#*) ;;     # ignore any hash signs
    *) echo $LINE
      ;;
    esac
  done <$INPUT_FILE
else
  echo "`basename $0` : Sorry $INPUT_FILE does not exist or is empty"
  exit 1
fi
```

运行得：

```
$ ignore_hash
AUDITSCM=full
AUDITSUB=/usr/opt/audit/sub
HASHSER=12890AB3
```

18.7.9 处理格式化报表

读报表文件时，一个常用任务是将不想要的行剔除。以下是库存商品水平列表，我们感兴趣的是那些包含商品记录当前水平的列

```
$ pg order
#########  RE-ORDER REPORT #########
 ITEM                 ORDERLEVEL      LEVEL
#####################################
Pens                  14              12
Pencils               15              15
Pads                   7               3
Disks                  3               2
Sharpeners             5               1
######################################
```

我们的任务是读取其中取值，决定哪些商品应重排。如果重排，重排水平应为现在商品

China-pub.com 下载

的两倍。输出应打印需要重排的每种商品数量及重排总数。

我们已经知道可以忽略以某些字符开始的行，因此这里没有问题。首先读文件，忽略所有注释行和以‘ ITEM ’开始的标注行。读取文件至一临时工作文件中，为确保不存在空行，用sed删除空行，需要真正做的是过滤文本文件。脚本如下：

```
    # use to sed to delete any empty lines, if any
    sed -e '/^$/d' order.tmp >order.$$
    mv order.$$ order.tmp
  else
    echo "`basename $0` : Sorry $INPUT_FILE does not exist or is empty"
  fi
```

输出为：

```
$ pg order.tmp
Pens        14 12
Pencils     15 15
Pads         7  3
Disks        3  2
Sharpeners   5  1
```

现在要在另一个while循环中读取临时工作文件，使用 expr对数字进行数值运算。

```
$ pg whileorder2
#!/bin/sh
# whileorder2
# init the variables
HOLD=order.tmp
RE_ORDER=0
ORDERS=0
STATIONERY_TOT=0
if [ -s $HOLD ]; then
  echo "=========== STOCK RE_ORDER REPORT ==============="
  while read ITEM REORD LEVEL
  do
    # are we below the reorder level for this item??
```

China-pub.com 下载

```
    if [ "$LEVEL" -lt "$REORD" ]; then
      # yes, do the new order amount
      NEW_ORDER=`expr $REORD + $REORD`
      # running total of orders
      ORDERS=`expr $ORDERS + 1`
      # running total of stock levels
      STATIONERY_TOT=`expr $STATIONERY_TOT + $LEVEL`
      echo "$ITEM need reordering to the amount $NEW_ORDER"
    fi
  done <$HOLD
  echo "$ORDERS new items need to be ordered"
  echo "Our reorder total is $STATIONERY_TOT"
else
else
  echo "`basename $0` : Sorry $HOLD does not exist or is empty"
fi
```

以下为依据报表文件运行所得输出结果。

```
$ whileorder
=========== STOCK REORDER REPORT ===============
Pens need reordering to the amount 28
Pads need reordering to the amount 14
Disks need reordering to the amount 6
Sharpeners need reordering to the amount 10
4 new items need to be ordered
Our reorder total is 18
```

将两段脚本结合在一起很容易。实际上这本来是一个脚本，为讲解方便，才将其分成两个。

18.7.10 while循环和文件描述符

第5章查看文件描述符时，提到有必要用while循环将数据读入一个文件。使用文件描述符3和4，下面的脚本进行文件myfile.txt到myfile.bak的备份。注意，脚本开始测试文件是否存在，如果不存在或没有数据，脚本立即终止。还有while循环用到了空命令（ :），这是一个死循环，因为null永远返回真。尝试读至文件结尾将返回错误，那时脚本也终止执行。

```
$ pg copyfile
#!/bin/sh
# copyfile
FILENAME=myfile.txt
FILENAME_BAK=myfile.bak
if [ -s $FILENAME ]; then
  # open FILENAME for writing
  # open FILENAME for reading
  exec 4>$FILENAME_BAK
  exec 3<$FILENAME

# loop forever until no more data and thus an error so we are at end of
file
  while :
  do
    read LINE <&3
    if [ "$?" -ne 0 ]; then
      # errors then close up
```

```
        exec 3<&-
        exec 4<&-
        exit
      fi
    # write to FILENAME_BAK
    echo $LINE>&4
    done
else
    echo "`basename $0` : Sorry, $FILENAME is not present or is empty" >&2
fi
```

18.8 使用break和continue控制循环

有时需要基于某些准则退出循环或跳过循环步。 shell提供两个命令实现此功能。

• break。

• continue。

18.8.1 break

break命令允许跳出循环。 break通常在进行一些处理后退出循环或 case语句。如果是在一个嵌入循环里，可以指定跳出的循环个数。例如如果在两层循环内，用 break 2刚好跳出整个循环。

18.8.2 跳出case语句

下面的例子中，脚本进入死循环直至用户输入数字大于 5。要跳出这个循环，返回到 shell提示符下， break使用脚本如下：

```
$ pg breakout
#!/bin/sh
# breakout
# while : means loop forever
while :
do
  echo -n "Enter any number [1..5] :"
  read ANS
  case $ANS in
    1|2|3|4|5) echo "great you entered a number between 1 and 5"
    ;;
  *) echo "Wrong number..bye"
    break
    ;;
  esac
done
```

18.8.3 continue

continue命令类似于break命令，只有一点重要差别，它不会跳出循环，只是跳过这个循环步。

China-pub.com 下载

18.8.4 浏览文件行

下面是一个前面用过的人人文件列表，但是现在加入了一些头信息。

```
$ pg names2.txt
------LISTING OF PERSONNEL FILE-----
--- TAKEN AS AT 06/1999 -----
Louise Conrad:Accounts:ACC8987
Peter James:Payroll:PR489
Fred Terms:Customer:CUS012
James Lenod:Accounts:ACC887
Frank Pavely:Payroll:PR489
```

假定现在需要处理此文件，看过文件之后知道头两行并不包含个人信息，因此需要跳过这两行。

也不需要处理雇员Peter James，这个人已经离开公司，但没有从人员文件中删除。

对于头信息。只需简单计算所读行数，当行数大于 2时开始处理，如果处理人员名字为Peter James，也将跳过此记录。脚本如下：

```
$ pg whilecontinue
#!/bin/sh
# whilecontinue
SAVEDIFS=$IFS
IFS=:
INPUT_FILE=names2.txt
NAME_HOLD="Peter James"
LINE_NO=0

if [ -s $INPUT_FILE ]; then
  while read NAME DEPT ID
  do
    LINE_NO=`expr $LINE_NO + 1`
    if [ "$LINE_NO" -le 2 ]; then
      # skip if the count is less than 2
      continue
    fi
    if [ "$NAME" = "$NAME_HOLD" ]; then
      # skip if the name in NAME_HOLD is Peter James
      continue
    else
      echo " Now processing...$NAME $DEPT $ID"
      # all the processing goes here
    fi
  done < $INPUT_FILE
IFS=$SAVEDIFS
else
  echo "`basename $0 ` : Sorry file not found or there is no data in the
file" >&2
  exit 1
fi
```

运行上面脚本，得出：

```
$ whilecontinue
Louise Conrad Accounts ACC8987
Fred Terms Customer CUS012
```

```
James Lenod Accounts ACC887
Frank Pavely Payroll PR489
```

18.9 菜单

创建菜单时，在while循环里null空命令很合适。 hile加空命令null意即无限循环，这正是一个菜单所具有的特性。当然除非用户选择退出或是一个有效选项。

创建菜单只需用while循环和case语句捕获用户输入的所有模式。如果输入无效，则报警，反馈错误信息，然后继续执行循环直到用户完成处理过程，选择退出选项。

菜单界面应是友好的，不应该让用户去猜做什么，主屏幕也应该带有主机名和日期，并伴随有运行此菜单的用户名。由于测试原因，所有选项使用的是系统命令。

下面是即将显示的菜单。

```
-----------------------------------------------------------------
User: dave              Host:Bumper             Date:31/05/1999
-----------------------------------------------------------------
                1 :  List files in current directory
                2 :  Use the vi editor
                3 :  See who is on the system
                H :  Help screen
                Q :  Exit Menu
-----------------------------------------------------------------
              Your Choice [1,2,3,H,Q] >
```

首先，使用命令替换设置日期，主机名和用户。日期格式为 /DD/MM/YYYY，参数格式为：

```
$ date +%d/%m/%y
32/05/1999
```

对于主机名，使用hostname-s选项只抽取主机名部分。主机名有时也包括了完全确认的域名。当然如果要在屏幕上显示这些，那就更好了。

可以给变量一个更有意义的名字：

```
MYDATE=`date +%d/%m/%Y`
THIS_HOST=`hostname -s`
USER=`whoami`
```

对于while循环，只需将空命令直接放在while后，即为无限循环，格式为：

```
while :
do
    命令
done
```

要注意实际屏幕显示，不要浪费时间使用大量的 echo语句或不断地调整它们。这里使用本地文档，在分界符后面接受输入，直至分界符被再次定位。格式为：

```
command <<WORD
any input
WORD
```

此技术用于菜单屏幕，也将用于帮助屏幕。帮助屏幕不像这样复杂。

用case语句控制用户选择。菜单选择有：

China-pub.com 下载

1 : List files in current directory
2 : Use the vi editor
3 : See who is on the system
H: Help screen
Q: Exit Menu

case语句应控制所有这些模式，不要忘了将大写与小写模式并列在一起，因为有时用户会关闭或打开CAPS LOCK键。因为菜单脚本不断循环，所以应该允许用户退出，如果用户选择Q或q键，脚本应退出，此时脚本带有0值。

如果用户选择无效，应发出警报并带有警告信息。虽然本章开始说过从现在开始一直使用LINUX或BSD echo语句版本，这里必须使用系统V版本发出警报：

```
echo "\007 the bell ring"
```

用一个简单的echo和读语句锁屏直到用户点击回车键，这样任何信息或命令输出将可视。

也需要清屏，为此可使用tput命令（后面讨论tput），如果不这样做，使用clear命令也可以。到此所有功能已经具备了，脚本如下：

```
$ pg menu
#!/bin/sh
# menu
# set the date, user and hostname up
MYDATE=`date +%d/%m/%Y`
THIS_HOST=`hostname -s`
USER=`whoami`
# loop forever !
while :
do
  # clear the screen
  tput clear
  # here documents starts here
  cat <<MAYDAY
  ------------------------------------------------------------------
  User: $USER              Host:$THIS_HOST              Date:$MYDATE
  ------------------------------------------------------------------
                  1 :  List files in current directory
                  2 :  Use the vi editor
                  3 :  See who is on the system
                  H :  Help screen
                  Q :  Exit Menu
  ------------------------------------------------------------------
  MAYDAY

  # here document finished
  echo -e -n "\tYour Choice [1,2,3,H,Q] >"
  read CHOICE
    case $CHOICE in
    1) ls
      ;;
    2) vi
      ;;
    3) who
      ;;
    H|h)
```

China-pub.com
下载


```
        # use a here document for the help screen
        cat <<MAYDAY
        This is the help screen, nothing here yet to help you!
        MAYDAY
          ;;
        Q|q) exit 0
          ;;
        *) echo -e "\t\007unknown user response"
          ;;
        esac
      echo -e -n "\tHit the return key to continue"
      read DUMMY
    done
```

18.10 小结

在任何合理脚本的核心部分都有某种流控制，如果要求脚本具有智能性，必须能够进行判断和选择。

本章讲述了怎样使用控制流写一段优美的脚本，而不只是完成基本功能。这里也学到了怎样处理列表和循环直至条件为真或为假。

China-pub.com 下载

第19章 shell 函 数

本书目前为止所有脚本都是从头到尾执行。这样做很好，但你也许已经注意到有些脚本段间互相重复。

shell允许将一组命令集或语句形成一个可用块，这些块称为 shell函数。

本章内容有：

- 定义函数。
- 在脚本中使用函数。
- 在函数文件中使用函数。
- 函数举例。

函数由两部分组成：

函数标题。

函数体。

标题是函数名。函数体是函数内的命令集合。标题名应该唯一；如果不是，将会混淆结果，因为脚本在查看调用脚本前将首先搜索函数调用相应的 shell。

定义函数的格式为：

```
函数名 ()
{
命令1
 ...
}
```

或者

```
函数名 () {
命令1
...
}
```

两者方式都可行。如果愿意，可在函数名前加上关键字 function，这取决于使用者。

```
function   函数名 ()
{   ...
}
```

可以将函数看作是脚本中的一段代码，但是有一个主要区别。执行函数时，它保留当前shell和内存信息。此外如果执行或调用一个脚本文件中的另一段代码，将创建一个单独的shell，因而去除所有原脚本中定义的存在变量。

函数可以放在同一个文件中作为一段代码，也可以放在只包含函数的单独文件中。函数不必包含很多语句或命令，甚至可以只包含一个 echo语句，这取决于使用者。

19.1 在脚本中定义函数

以下是一个简单函数

```
hello ()
{
echo "Hello there today's date is `date`"
}
```

所有函数在使用前必须定义。这意味着必须将函数放在脚本开始部分，直至 shell解释器首次发现它时，才可以使用。调用函数仅使用其函数名即可。上面的例子中，函数名为 hello，函数体包含一个echo语句，反馈当天日期。

19.2 在脚本中使用函数

现在创建函数，观察其在脚本中的用法。

```
$ pg func1
#!/bin/sh
# func1
hello ()
{
echo "Hello there today's date is `date`"
}

echo "now going to the function hello"
hello

echo "back from the function"
```

运行脚本，结果为：

```
$ func1
now going to the function hello
Hello there today's date is Sun Jun  6 10:46:59 GMT 1999
back from the function
```

上面例子中，函数定义于脚本顶部。可以在脚本中使用函数名 hello调用它。函数执行后，控制返回函数调用的下一条语句，即反馈语句 back from the function。

19.3 向函数传递参数

向函数传递参数就像在一般脚本中使用特殊变量 $1,$2...$9一样，函数取得所传参数后，将原始参数传回 shell脚本，因此最好先在函数内重新设置变量保存所传的参数。这样如果函数有一点错误，就可以通过已经本地化的变量名迅速加以跟踪。函数里调用参数（变量）的转换以下划线开始，后加变量名，如：_FILENAME或_filename。

19.4 从调用函数中返回

当函数完成处理或希望函数基于某一测试语句返回时，可做两种处理：

1) 让函数正常执行到函数末尾，然后返回脚本中调用函数的控制部分。

2) 使用return返回脚本中函数调用的下一条语句，可以带返回值。0为无错误，1为有错误。这是可选的，与最后状态命令报表例子极其类似。其格式为：

China-pub.com 下载

```
return     从函数中返回，  用最后状态命令决定返回值。
Return 0   无错误返回。
Return 1   有错误返回
```

19.5 函数返回值测试

可以直接在脚本调用函数语句的后面使用最后状态命令来测试函数调用的返回值。例如：

```
check_it_is_a_directory $FILENAME    # this is the function call and check
if [ $? = 0 ]    # use the last status command now to test
then
  echo "All is OK"
else
  echo " Something went wrong!"
fi
```

更好的办法是使用if语句测试返回0或者返回1。最好在if语句里用括号将函数调用括起来以增加可读性。例如：

```
if check_it_is_a_directory $FILENAME; then
  echo "All is OK"
  # do something ??
else
  echo "Something went wrong !"
  # do something ??
fi
```

如果函数将从测试结果中反馈输出，那么使用替换命令可保存结果。函数调用的替换格式为：

```
variable_name=function_name
```

函数function_name输出被设置到变量variable_name中。

不久我们会接触到许多不同的函数及使用函数的返回值和输出的不同方法。

19.6 在shell中使用函数

当你收集一些经常使用的函数时，可以将之放入函数文件中并将文件载入 shell。

文件头应包含语句#!/bin/sh，文件名可任意选取，但最好与相关任务有某种实际联系。例如，functions.main。

一旦文件载入shell，就可以在命令行或脚本中调用函数。可以使用 set命令查看所有定义的函数。输出列表包括已经载入shell的所有函数。

如果要改动函数，首先用unset命令从shell中删除函数，尽管unset删除了函数以便于此函数对于shell或脚本不可利用，但并不是真正的删除。改动完毕后，再重新载入此文件。有些shell会识别改动，不必使用unset命令，但为了安全起见，改动函数时最好使用unset命令。

19.7 创建函数文件

下面创建包容函数的函数文件并将之载入shell，进行测试，再做改动，之后再重新载入。

函数文件名为functions.main，内容如下：

China-pub.com
下载


```
$ pg functions.main
#!/bin/sh
# functions.main
#
# findit: this is front end for the basic find command
findit() {
# findit
if [ $# -lt 1 ]; then
  echo "usage :findit file"
  return 1
fi
find / -name $1 -print
```

上述脚本本书前面用过，现在将之转化为一个函数。这是一个基本 find命令的前端。如果不加参数，函数将返回 1，即发生错误。注意错误语句中用到了实际函数名，因为这里用 $0，shell将只返回sh-信息，原因是文件并不是一个脚本文件。这类信息对用户帮助不大。

19.8 定位文件

定位文件格式为：

```
./pathname/filename
```

现在文件已经创建好了，要将之载入 shell，试键入：

```
$. functions.main
```

如果返回信息file not found，再试：

```
$. /functions.main
```

此即<点><空格><斜线><文件名>，现在文件应该已载入 shell。如果仍有错误，则应该仔细检查是否键入了完整路径名。

19.9 检查载入函数

使用set命令确保函数已载入。 set命令将在shell中显示所有的载入函数。

```
$ set
USER=dave
findit=()
{
if [ $# -lt 1 ]; then
  echo "usage :findit file";
  return 1;
fi;
find / -name $1 -print
}
...
```

19.10 执行shell函数

要执行函数，简单地键入函数名即可。这里是带有一个参数的 findit函数，参数是某个系统文件。

```
$ findit groups
```

```
/usr/bin/groups
/usr/local/backups/groups.bak
```

19.10.1 删除shell函数

现在对函数做一些改动。首先删除函数，使其对 shell不可利用。使用unset命令完成此功能。删除函数时unset命令格式为：

```
unset  function_name
$ unset findit
```

如果现在键入set命令，函数将不再显示。

19.10.2 编辑shell函数

编辑函数functions.main，加入for循环以便脚本可以从命令行中读取多个参数。改动后函数脚本如下：

```
$ pg functions.main
#!/bin/sh
findit()
{
# findit
if [ $# -lt 1 ]; then
  echo "usage :findit file"
  return 1
fi
for loop
do
  find / -name $loop -print
done
}
```

再次定位函数

```
$. /functions.main
```

使用set命令查看其是否被载入，可以发现shell正确解释for循环以接受所有输入参数。

```
$ set
findit=()
{
if [ $# -lt 1 ]; then
  echo "usage :`basename $0` file";
  return 1;
fi;
for loop in "$@";
do
  find / -name $loop -print;
done
}
...
```

现在执行改动过的findit函数，输入两个参数：

```
$ findit LPSO.doc passwd
/usr/local/accounts/LPSO.doc
/etc/passwd
...
```

19.10.3 函数举例

既然已经学习了函数的基本用法，现在就用它来做一些工作。函数可以节省大量的编程时间，因为它是可重用的。

1. 变量输入

以下脚本询问名，然后是姓。

```
$ pg func2
#!/bin/sh
# func2
echo -n "What is your first name :"
read F_NAME
echo -n "What is your surname :"
read S_NAME
```

要求输入字符必须只包含字母。如果不用函数实现这一点，要写大量脚本。使用函数可以将重复脚本删去。这里用awk语言测试字符。以下是取得只有小写或大写字符的测试函数。

```
char_name()
{
# char_name
# to call: char_name string
# assign the argument across to new variable
_LETTERS_ONLY=$1
# use awk to test for characters only !
_LETTERS_ONLY=`echo $1|awk '{if($0~/[^a-z A-Z]/) print "1"}'`
if [ "$_LETTERS_ONLY" != "" ]
then
  # oops errors
  return 1
else
  # contains only chars
  return 0
fi
}
```

首先设置变量$1为一有意义的名字，然后用awk测试整个传送记录只包含字母，此命令输出（1为非字母，空为成功）保存在变量_LETTERS_ONLY中。

然后执行变量测试，如果为空，则为成功，如果有值，则为错误。基于此项测试，返回码然后被执行。在对脚本的函数调用部分进行测试时，使用返回值会使脚本清晰易懂。

使用if语句格式测试函数功能：

```
if char_name $F_NAME; then
  echo "OK"
else
  echo "ERRORS"
fi
```

如果有错误，可编写一个函数将错误反馈到屏幕上。

```
name_error()
# name_error
# display an error message
{
echo " $@ contains errors, it must contain only letters"
}
```

China-pub.com 下载

函数name_error用于显示所有无效输入错误。使用特殊变量 $@显示所有参数，这里为变量F_NAME和S_NAME值。完成脚本如下：

```
$ pg func2
!/bin/sh
char_name()
# char_name
# to call: char_name string
# check if $1 does indeed contain only characters a-z,A-Z
{
# assign the argurment across to new variable
_LETTERS_ONLY=$1
_LETTERS_ONLY=`echo $1|awk '{if($0~/[^a-zA-Z]/) print "1"}'`
if [ "$_LETTERS_ONLY" != "" ]
then
  # oops errors
  return 1
else
  # contains only chars
  return 0
fi
}

name_error()
# display an error message
{
echo " $@ contains errors, it must contain only letters"
}

while :
do
  echo -n "What is your first name :"
  read F_NAME
  if char_name $F_NAME
  then
    # all ok breakout
    break
  else
    name_error $F_NAME
  fi
done

while :
do
  echo -n "What is your surname :"
  read S_NAME
  if char_name $S_NAME
  then
    # all ok breakout
    break
  else
    name_error $S_NAME
  fi
done
```

注意每个输入的while循环，这将确保不断提示输入直至为正确值，然后跳出循环。当然，

实际脚本拥有允许用户退出循环的选项，可使用适当的游标，正像控制 0长度域一样。

```
$ func2
What is your first name :Davi2d
 Davi2d contains errors, it must contain only letters
What is your first name :David
What is your surname :Tansley1
Tansley1 contains errors, it must contain only letters
What is your surname :Tansley
```

2. echo问题

echo语句的使用类型依赖于使用的系统是LINUX、BSD还是系统V，本书对此进行了讲解。下面创建一个函数决定使用哪种echo语句。

使用echo时，提示应放在语句末尾，以等待从read命令中接受进一步输入。

LINUX和BSD为此使用echo命令-n选项。

以下是LINUX（BSD）echo语句实例，这里提示放于echo后面：

```
$ echo -n "Your name :"
Your name : {{?}}
```

系统V使用\C保证在末尾提示：

```
$ echo "Your name :\c"
Your name : □
```

在echo语句开头LINUX使用-e选项反馈控制字符。其他系统使用反斜线保证 shell获知控制字符的存在。

有两种方法测试echo语句类型，下面讲述这两种方法，这样，就可以选择使用其中一个。

第一种方法是在echo语句里包含测试控制字符。如果键入\007和一个警铃，表明为系统V，如果只键入\007，显示为LINUX。

以下为第一个控制字符测试函数。

```
uni_prompt ()
# uni_prompt
# universal echo
{
if [ `echo "\007"` = "\007" ] >/dev/null 2>&1
# does a bell sound or are the characters just echoed??
then
  # characters echoed, it's LINUX /BSD
  echo -e -n "$@"
else
  # it's System V
  echo "$@\c"
fi
}
```

注意这里又用到了特殊变量$@以反馈字符串，要在脚本中调用上述函数，可以使用：

uni_prompt "\007 there goes the bell ,What is your name:"

这将发出警报并反馈‘ What is your name: ’，并在行尾显示字符串。如果在末尾出现字符，则为系统V版本，否则为LINUX/BSD版本。

第二种方法使用系统V \c测试字母z是否悬于行尾。

```
uni_prompt ()
```

China-pub.com 下载

```
# uni_prompt
# universal prompt
{
if [ `echo "Z\c"` = "Z" ] >/dev/null 2>&1
# echo any chracter out, does it hang on to the end of line ???
then
  # yes, it's System V
  echo "$@\c"
else
  # No, it's LINUX, BSD
  echo -e -n "$@"
fi
}
```

要在脚本中调用上述函数，可以使用：

```
uni_prompts "\007 there goes the,bell What is your name:"
```

使用两个函数中任意一个，并加入一小段脚本：

```
uni_prompt "\007 There goes the bell, What is your name :"
read NAME
```

将产生下列输出：

```
There goes the bell What is your name:
```

3. 读单个字符

在菜单中进行选择时，最麻烦的工作是必须在选择后键入回车键，或显示“ press any key to continue ”。可以使用dd命令解决不键入回车符以发送击键序列的问题。

dd命令常用于对磁带或一般的磁带解压任务中出现的数据问题提出质疑或转换，但也可用于创建定长文件。下面创建长度为1兆的文件myfile。

```
dd if:/dev/zero of=myfile count=512 bs=2048
```

dd命令可以翻译键盘输入，可被用来接受多个字符。这里如果只要一个字符， dd命令需要删除换行字符，这与用户点击回车键相对应。 dd只送回车前一个字符。在输入前必须使用stty命令将终端设置成未加工模式，并在dd执行前保存设置，在dd完成后恢复终端设置。

函数如下：

```
read_a_char()
# read_a_char
{
# save the settings
SAVEDSTTY=`stty -g`
# set terminal raw please
   stty cbreak
  # read and output only one character
   dd if=/dev/tty bs=1 count=1 2> /dev/null
  # restore terminal and restore stty
   stty -cbreak
stty $SAVEDSTTY
}
```

要调用函数，返回键入字符，可以使用命令替换操作，例子如下：

```
echo  -n "Hit Any Key To Continue"
character=`read_a_char`
echo " In case you are wondering you pressed $character"
```

4. 测试目录存在

拷贝文件时，测试目录是否存在是常见的工作之一。以下函数测试传递给函数的文件名是否是一个目录。因为此函数返回时带有成功或失败取值，可用 if语句测试结果。

函数如下：

```
is_it_a_directory()
{
# is_it_a_directory
# to call: is_it_a_directory directory_name
if [ $# -lt 1 ]; then
  echo "is_it_a_directory: I need an argument"
  return 1
fi
# is it a directory ?
_DIRECTORY_NAME=$1
if [ ! -d $_DIRECTORY_NAME ]; then
  # no it is not
  return 1
else
  # yes it is
  return 0
fi
}
```

要调用函数并测试结果，可以使用：

```
echo -n "enter destination directory :"
read DIREC
if is_it_a_directory $direc;
then :
else
  echo "$DIREC does not exist, create it now ? [y..n]"
  # commands go here to either create the directory or exit
  ...
  ...
fi
```

5. 提示Y或N

许多脚本在继续处理前会发出提示。大约可以提示以下动作：

• 创建一个目录。

• 是否删除文件。

• 是否后台运行。

• 确认保存记录。

等等

以下函数是一个真正的提示函数，提供了显示信息及缺省回答方式。缺省回答即用户按下回车键时采取的动作。case语句用于捕获回答。

```
continue_prompt()
# continue prompt
```

China-pub.com 下载

```
# check we have the right params
if [ $# -lt 1 ]; then
  echo "continue_prompt: I need a string to display"
  return 1
fi
# loop forever
while :
do
  echo -n "$_STR [Y..N] [$_DEFAULT]:"
  read _ANS
  # if user hits return set the default and determine the return value,
  # that's a : then a <space> then $
  :  ${_ANS:=$_DEFAULT}
  if [ "$_ANS" = "" ]; then
    case $_ANS in
    Y) return 0 ;;
    N) return 1 ;;
    esac
  fi
  # user has selected something
  case $_ANS in
  y|Y|Yes|YES)
    return 0
    ;;
  n|N|No|NO)
    return 1
    ;;
  *) echo "Answer either Y or N, default is $_DEFAULT"
    ;;
  esac
  echo $_ANS
done
}
```

要调用上述函数，须给出显示信息或参数 $1，或字符串变量。缺省回答 Y或N方式也必须指定。

以下是几种函数continue_prompt的调用格式。

```
if continue_prompt "Do you want to delete the var filesytem" "N"; then
  echo "Are you nuts!!"
else
  echo "Phew !, what a good answer"
fi
```

在脚本中加入上述语句，给出下列输入：

```
Do you really want to delete the var filesystem [Y..N] [N]:
phew !!

Do you really want to delete the var filesystem [Y..N] [N]:y
are you nuts..
```

现在可以看出为什么函数要有指定的缺省回答。

以下是函数调用的另一种方式：

```
if continue_prompt "Do you really want to print this report" "Y"; then
  lpr report
else:
```

```
fi
```

也可以使用字符串变量$1调用此函数：

```
if continue_prompt $1 "Y"; then
  lpr report
else:
fi
```

6. 从登录ID号中抽取信息

当所在系统很庞大，要和一登录用户通信时，如果忘了用户的全名，这是很讨厌的事。比如有时你看到用户锁住了一个进程，但是它们的用户 ID号对你来说没有意义，因此必须要用grep passwd文件以取得用户全名，然后从中抽取可用信息，向其发信号，让其他用户开锁。以下函数用于从grep /etc/passwd命令抽取用户全名。

本系统用户全名位于passwd文件域5中，用户的系统可能不是这样，这时必须改变其域号以匹配passwd文件。

这个函数需要一个或多个用户 ID号作为参数。它对密码文件进行 grep操作。

函数脚本如下：

```
whois()
# whois
# to call: whois userid
{
# check we have the right params
if [ $# -lt 1 ]; then
  echo "whois : need user id's please"
  return 1
fi

for loop
do
  _USER_NAME=`grep $loop /etc/passwd | awk -F: '{print $4}'`
  if [ "$_USER_NAME" = "" ]; then
    echo "whois: Sorry cannot find $loop"
  else
    echo "$loop is $_USER_NAME"
  fi
done
}
```

以下为whois函数调用方式：

```
$ whois dave peters superman
dave is David Tansley - admin accts
peter is Peter Stromer - customer services
whois: Sorry cannot find superman
```

7. 列出文本文件行号

在vi编辑器中，可以列出行号来进行调试，但是如果打印几个带有行号的文件，必须使用nl命令。以下函数用nl命令列出文件行号。原始文件中并不带有行号。

```
number_file()
# number_file
# to call: number_file filename
{
_FILENAME=$1
```

China-pub.com 下载

```
# check we have the right params
if [ $# -ne 1 ]; then
  echo "number_file: I need a filename to number"
  return 1
fi

loop=1
  while read LINE
  do
    echo "$loop: $LINE"
    loop=`expr $loop + 1`
  done < $_FILENAME
}
```

要调用number_file函数，可用一个文件名做参数，或在shell中提供一文件名，例如：

```
$ number_file myfile
```

也可以在脚本中这样写或用：

```
number_file $1
```

输出如下：

```
$ number_file /home/dave/file_listing
1: total 105
2: -rw-r--r--   1 dave     admin      0 Jun  6 20:03 DT
3: -rw-r--r--   1 dave     admin    306 May 23 16:00 LPSO.AKS
4: -rw-r--r--   1 dave     admin    306 May 23 16:00 LPSO.AKS.UC
5: -rw-r--r--   1 dave     admin    324 May 23 16:00 LPSO.MBB
6: -rw-r--r--   1 dave     admin    324 May 23 16:00 LPSO.MBB.UC
7: -rw-r--r--   1 dave     admin    315 May 23 16:00 LPSO.MKQ
...
...
```

8. 字符串大写

有时需要在文件中将字符串转为大写，例如在文件系统中只用大写字符创建目录或在有效的文本域中将输入转换为大写数据。

以下是相应功能函数，可以想像要用到tr命令：

```
str_to_upper ()
# str_to_upper
# to call: str_to_upper $1
{
_STR=$1
# check we have the right params
if [ $# -ne 1 ]; then
  echo "number_file: I need a string to convert please"
  return 1
fi
echo $@ |tr '[a-z]' '[A-Z]'
}
```

变量upper保存返回的大写字符串，注意这里用到特定参数 $@来传递所有参数。str_to_upper可以以两种方式调用。在脚本中可以这样指定字符串。

```
UPPER=`str_to_upper "documents.live"`
echo $upper
```

或者以函数输入参数$1的形式调用它。

```
UPPER=`str_to_upper $1`
echo $UPPER
```

两种方法均可用替换操作以取得函数返回值。

9. is_upper

虽然函数str_to_upper做字符串转换，但有时在进一步处理前只需知道字符串是否为大写。is_upper实现此功能。在脚本中使用if语句决定传递的字符串是否为大写。

函数如下：

```
is_upper()
# is_upper
# to call: is_upper $1
{
# check we have the right params
if [ $# -ne 1 ]; then
  echo "is_upper: I need a string to test OK"
  return 1
fi
# use awk to check we have only upper case
_IS_UPPER=`echo $1|awk '{if($0~/[^A-Z]/) print "1"}'`
if [ "$_IS_UPPER" != "" ]
then
  # no, they are not all upper case
  return 1
else
  # yes all upper case
  return 0
fi
}
```

要调用is_upper，只需给出字符串参数。以下为其调用方式：

```
echo -n "Enter the filename :"
read  FILENAME
if is_upper $FILENAME; then
  echo "Great it's upper case"
  # let's create a file  maybe ??
else
  echo "Sorry it's not upper case"
  # shall we convert it anyway using str_to_upper ???
fi
```

要测试字符串是否为小写，只需在函数 is_upper中替换相应的 awk语句即可。此为is_lower。

```
_IS_LOWER=`echo $1|awk '{if($0~/[^a-z]/) print "1"}'`
```

10. 字符串小写

现在实现此功能，因为已经给出了 str_to_upper，最好相应给出str_to_lower。函数工作方式与前面一样。

函数如下：

```
str_to_lower ()
# str_to_lower
# to call: str_to_lower $1
{
```

```
# check we have the right params
if [ $# -ne 1 ]; then
  echo "str_to_lower: I need a string to convert please"
  return 1
fi
echo $@ |tr '[A-Z]' '[a-z]'
}
```

变量LOWER保存最近返回的小写字符串。注意用到特定参数 $@传递所有参数。str_to_lower调用方式也分为两种。可以在脚本中给出字符串：

```
LOWER=`str_to_lower "documents.live"`
echo $LOWER
```

或在函数中用参数代替字符串：

```
LOWER=`str_to_upper $1`
echo $LOWER
```

11. 字符串长度

在脚本中确认域输入有效是常见的任务之一。确认有效包括许多方式，如输入是否为数字或字符；域的格式与长度是否为确定形式或值。

假定脚本要求用户交互输入数据到名称域，你会想控制此域包含字符数目，比如人名最多为20个字符。有可能用户输入超过 50个字符。以下函数实施控制功能。需要向函数传递两个参数，实际字符串和字符串最大长度。

函数如下：

```
check_length()
# check_length
# to call: check_length string max_length_of_string
{
_STR=$1
_MAX=$2
# check we have the right params
if [ $# -ne 2 ]; then
  echo "check_length: I need a string and max length the string should be"
  return 1
fi
# check the length of the string
_LENGTH=`echo $_STR |awk '{print length($0)}'`
if [ "$_LENGTH" -gt  "$_MAX" ]; then
  # length of string is too big
  return 1
else
  # string is ok in length
  return 0
fi
}
```

调用函数check_length：

```
$ pg test_name
# !/bin/sh
# test_name
while :
do
```

```
    echo -n "Enter your FIRST name :"
    read NAME
    if check_length $NAME 10
    then
      break
      # do nothing fall through condition all is ok
    else
      echo "The name field is too long 10 characters max"
    fi
done
```

循环持续直到输入到变量NAME的数据小于最大字符长度，这里指定为10，break命令然后跳出循环。

使用上述脚本段，输出结果如下：

```
$ val_max
Enter your FIRST name :Perterrrrrrrrrrrrrrr
The name field is too long 10 characters max
Enter your FIRST name :Peter
```

可以使用wc命令取得字符串长度。但是要注意，使用wc命令接受键盘输入时有一个误操作。如果用户输入了一个名字后，点击了几次空格键， wc会将这些空格也作为字符串的一部分，因而给出其错误长度。awk在读取键盘时缺省截去字符串末尾处空格。

以下是wc命令的缺点举例（也可以称为特征之一）

```
echo -n "name :"
read NAME
echo $NAME | wc -c
```

运行上述脚本（其中□为空格）

```
 name :Peter□□
       6
```

12. chop

chop函数删除字符串前面字符。可以指定从第一个字符起删去的字符数。假定有字符串MYDOCUMENT.DOC，要删去MYDUCUMENT部分，以便函数只返回.DOC，需要把下述命令传给chop函数：

```
MYDOCUMENT.DOC 10
```

Chop 函数如下：

```
chop()
# chop
# to call:chop string how_many_chars_to_chop
{
_STR=$1
_CHOP=$2
# awk's substr starts at 0, we need to increment by one
# to reflect when the user says (ie) 2 chars to be chopped it will be 2
chars off
# and not 1
CHOP=`expr $_CHOP + 1`

# check we have the right params
if [ $# -ne 2 ]; then
  echo "check_length: I need a string and how many characters to chop"
```

China-pub.com 下载

```
  return 1
fi
# check the length of the string first
# we can't chop more than what's in the string !!
_LENGTH=`echo $_STR |awk '{print length($0)}'`
if [ "$_LENGTH" -lt "$_CHOP" ]; then
  echo "Sorry you have asked to chop more characters than there are in
       the string"
  return 1
fi
echo $_STR |awk '{print substr($1,'$_CHOP')}'
}
```

删除后字符串保存于变量CHOPPED中，使用下面方法调用chop函数：

```
CHOPPED=`chop "Honeysuckle" 5`
echo $CHOPPED
suckle
```

或者：

```
echo -n "Enter the Filename :"
read FILENAME
CHOPPED=`chop $FILENAME 1`
# the first character would be chopped off !
```

13. MONTHS

产生报表或创建屏幕显示时，为方便起见有时要快速显示完整月份。函数 months，接受月份数字或月份缩写作为参数，返回完整月份。

例如，传递3或者03可返回March。函数如下：

```
months()
{
# months
_MONTH=$1
# check we have the right params
if [ $# -ne 1 ]; then
  echo "months: I  need a number 1 to 12 "
  return 1
fi

case $_MONTH in
1|01|Jan)_FULL="January" ;;
2|02|Feb)_FULL="February" ;;
3|03|Mar)_FULL="March";;
4|04|Apr)_FULL="April";;
5|05|May)_FULL="May";;
6|06|Jun)_FULL="June";;
7|07|Jul)_FULL="July";;
8|08|Aug)_FULL="August";;
9|10|Sep|Sept)_FULL="September";;
10|Oct)_FULL="October";;
11|Nov)_FULL="November";;
12|Dec)_FULL="December";;
*) echo "months: Unknown month"
  return 1
  ;;
```

```
esac
echo $_FULL
}
```

用下面方法调用函数months

```
months 04
```

上面例子显示April，脚本中使用：

```
MY_MONTH=`months 06`
echo "Generating the Report for Month End $MY_MONTH"
...
```

返回月份June。

19.10.4 将函数集中在一起

本章目前讲到的函数没有一定的顺序。这些例子只表明函数不一定很长或不一定为一些复杂的脚本。

本书许多函数脚本简单实用，并不需要任何新的后备知识。这些函数只是防止重复输入脚本，实际上这就是函数的基本功能。

本章开始部分，讲到怎样在 shell中使用函数。第一次使用函数时，也许要花一段时间才能理解其返回值的用法。

本章讲到了几种不同的调用函数及其返回值的方法。如果遇到问题，查看一下实例返回值及其测试方法即可。

以下是一些小技巧。测试函数时，首先将其作为代码测试，当结果满意时，再将其转换为函数，这样做可以节省大量的时间。

19.11 函数调用

本章最后讲述使用函数的两种不同方法：从原文件中调用函数和使用脚本中的函数。

19.11.1 在脚本中调用函数

要在脚本中调用函数，首先创建函数，并确保它位于调用之前。以下脚本使用了两个函数。此脚本前面提到过，它用于测试目录是否存在。

```
$ pg direc_check
!/bin/sh
# function file
is_it_a_directory()
{
# is_it_a_directory
# to call: is_it_a_directory directory_name
_DIRECTORY_NAME=$1
if [ $# -lt 1 ]; then
  echo "is_it_a_directory: I need a directory name to check"
  return 1
fi
# is it a directory ?
if [ ! -d $_DIRECTORY_NAME ]; then
  return 1
```

China-pub.com 下载

```
else
  return 0
fi
}
#-------------------------------------------------------
error_msg()
{
# error_msg
# beeps; display message; beeps again!
echo -e "\007"
echo $@
echo -e "\007"
  return 0
}
}

### END OF FUNCTIONS

echo -n "enter destination directory :"
read DIREC
if is_it_a_directory $DIREC
then :
else
  error_msg "$DIREC does not exist...creating it now"
  mkdir $DIREC > /dev/null 2>&1
  if [ $? != 0 ]
  then
    error_msg "Could not create directory:: check it out!"
    exit 1
  else :
  fi
fi # not a directory
echo "extracting files..."
```

上述脚本中，两个函数定义于脚本开始部分，并在脚本主体中调用。所有函数都应该在任何脚本主体前定义。注意错误信息语句，这里使用函数 error_msg显示错误，反馈所有传递到该函数的参数，并加两声警报。

19.11.2 从函数文件中调用函数

前面讲述了怎样在命令行中调用函数，这类函数通常用于系统报表功能。

现在再次使用上面的函数，但是这次将之放入函数文件 functions.sh里。sh意即shell脚本。

```
$ pg functions.sh
#!/bin/sh
# functions.sh
# main script functions
is_it_a_directory()
{
# is_it_a_directory
# to call: is_it_a_directory directory_name
#
if [ $# -lt 1 ]; then
  echo "is_it_a_directory: I need a directory name to check"
  return 1
```

China-pub.com 下载


```
fi
# is it a directory ?
DIRECTORY_NAME=$1
if [ ! -d $DIRECTORY_NAME ]; then
  return 1
else
  return 0
fi
}

#---------------------------------------------

error_msg()
{
echo -e "\007"
echo $@
echo -e "\007"
  return 0
}
```

现在编写脚本就可以调用 functions.sh中的函数了。注意函数文件在脚本中以下述命令格式定位：

```
.\<path to file>
```

使用这种方法不会创建另一个 shell，所有函数均在当前shell下执行。

```
$ pg direc_check
!/bin/sh
# direc_check
# source the function file functions.sh
# that's a <dot><space><forward slash>
. /home/dave/bin/functions.sh

# now we can use the function(s)

echo -n "enter destination directory :"
read DIREC
if is_it_a_directory $DIREC
then :
else
  error_msg "$DIREC does not exist...creating it now"
  mkdir $DIREC > /dev/null 2>&1
  if [ $? != 0 ]
  then
    error_msg "Could not create directory:: check it out!"
    exit 1
  else :
  fi
fi # not a directory
echo "extracting files..."
```

运行上述脚本，可得同样输出结果，好像函数在脚本中一样。

```
$ direc_check
enter destination directory :AUDIT
AUDIT does not exist...creating it now
extracting files...
```

China-pub.com 下载

19.12 定位文件不只用于函数

定位文件不只针对于函数，也包含组成配置文件的全局变量。

假定有两个备份文件备份同一系统的不同部分。最好让它们共享一个配置文件。为此需要在一个文件里创建用户变量，然后将一个备份脚本删除后，可以载入这些变量以获知用户在备份开始前是否要改变其缺省值。有时也许要备份到不同的媒体中。

当然这种方法可用于共享配置以执行某一过程的任何脚本。下面的例子中，配置文件backfunc包含一些备份脚本所共享的缺省环境。文件如下：

```
$ pg backfunc
#!/bin/sh
# name: backfunc
# config file that holds the defaults for the archive systems
_CODE="comet"
_FULLBACKUP="yes"
_LOGFILE="/logs/backup/"
_DEVICE="/dev/rmt/0n"
_INFORM="yes"
_PRINT_STATS="yes"
```

缺省文件很清楚，第1域_CODE包含一个脚本关键字。要查看并且改变缺省值，用户必须首先输入匹配_CODE取值的脚本，即“ comet ”。

以下脚本要求输入密码，成功后显示缺省配置。

```
$ pg readfunc
#!/bin/sh
# readfunc

if [ -r backfunc ]; then
  # source the file
  . /backfunc
else
  echo "$`basename $0` cannot locate backfunc file"
fi

echo -n "Enter the code name :"
# does the code entered match the code from backfunc file ???
if [ "${CODE}" != "${_CODE}" ]; then
  echo "Wrong code...exiting..will use defaults"
  exit 1
fi

echo " The environment config file reports"
echo "Full Backup Required             : $_FULLBACKUP"
echo "The Logfile Is                   : $_LOGFILE"
echo "The Device To Backup To is       : $_DEVICE"
echo "You Are To Be Informed by Mail   : $_INFORM"
echo "A Statistic Report To Be Printed: $_PRINT_STATS"
```

脚本运行时，首先要求输入脚本。脚本匹配后，可以查看缺省值。然后就可以编写脚本让用户改变缺省值。

```
$ readback
Enter the code name :comet
```

China-pub.com
下载


```
 The environment config file reports
Full Backup Required            : yes
The Logfile Is                  : /logs/backup/
The Device To Backup To is      : /dev/rmt/0n
You Are To Be Informed by Mail  : yes
A Statistic Report To Be Printed: yes
```

19.13 小结

使用函数可以节省大量的脚本编写时间。创建可用和可重用的脚本很有意义，可以使主脚本变短，结构更加清晰。

当创建了许多函数后，将之放入函数文件里，然后其他脚本就可以使用这些函数了。

China-pub.com
下载

第20章　向脚本传递参数

前面已经讲到如何使用特定变量 $1..$9向脚本传递参数。 $#用于统计传递参数的个数。可以创建一个usage语句，需要时可通知用户怎样以适当的调用参数调用脚本或函数。

本章内容有：

- shift。
- getopts。
- shift和getopts例子。

简单地说，下述脚本框架控制参数开始与停止。脚本需要两个参数，如果没有输入两个参数，那么产生一个usage语句。注意这里使用case语句处理输入脚本的不同参数。

```
$ pg opt
#!/bin/sh
# opt

usage()
{
echo "usage:‘basename $0‘ start|stop process name"
}

OPT=$1
PROCESSID=$1
if [ $# -ne 2 ]
then
  usage
  exit 1
fi
case $OPT in
start|Start) echo "Starting..$PROCESSID"
  #  some process to go here
  ;;
stop|Stop) echo "Stopping..$PROCESSID"
  # some process to go here
  ;;
*) usage
  ;;
;;
esac
```

执行脚本，输入以下参数，结果为：

```
$ opt start named
Starting..named

$ opt start
usage:opt start|stop process name
```

任何UNIX或LINUX命令均接受一般格式：

命令　选项　文件

China-pub.com 下载

选项部分最多可包含12个不同的值。上述脚本中，如果必须控制不同的命令选项，就要加入大量脚本。这里只控制两个选项：开始和停止。

幸运的是shell提供shift命令以帮助偏移选项，使用shift可以去除只使用$1到$9传递参数的限制。

20.1 shift命令

向脚本传递参数时，有时需要将每一个参数偏移以处理选项，这就是 shift命令的功能。它每次将参数位置向左偏移一位，下面用一段简单脚本详述其功能。脚本使用 while循环反馈所有传递到脚本的参数。

```
$ pg opt2
#!/bin/sh
# opt2
loop=0
while [ $# -ne 0 ]    # while there are still arguments
do
  echo $1
done
```

你可能想像，上述脚本一直执行，直到命令行中不再有更多的参数输入。错了，因为没有办法偏移到脚本中下一个参数，将只会反馈出第一个参数。执行结果如下：

```
$ opt2 file1 file2 file3
file1
file1
file1
...
```

20.1.1 shift命令简单用法

使用shift命令来处理传递到脚本的每一个参数。改动后脚本如下：

```
$ pg opt2
#!/bin/sh
# opt2
loop=0
while [ $# -ne 0 ]    # while there are still arguments
do
  echo $1
  shift
done
```

现在再执行，结果将会不同：

```
$ opt2 file1 file2 file3
file1
file2
file3
```

20.1.2 命令行输入的最后一个参数

虽然还没有讲eval命令，如果需要知道命令行中输入的最后一个参数（通常是一个文件名），可以有两种选择：使用命令eval echo \$$#；使用shift命令：shift 'expr $# -2'。

China-pub.com 下载

20.1.3 使用shift处理文件转换

shift可使控制命令行选项更加容易。下面构造一个转换脚本，使用 tr将文件名转换为大写或小写。

脚本选项为：

-l 用于小写转换。

-u 用于大写转换。

使用shift命令将脚本放在一起以控制 -l和-u选项。脚本的第一版本如下：

```
$ pg tr_case
!/bin/sh
# tr_case
# case conversion
usage()
{
# usage
echo "usage:`basename $0` -[l|u] file [files]" >&2
exit 1
}

if [ $# -eq 0 ]; then
  # no parameters passed !
  usage
fi

while [ $# -gt 0 ]
do
  case $1 in
  -u|-U) echo "-u option specified"
    # do any settings of variables here for lowercase then shift
    shift
    ;;
  -l|-L) echo "-l option specified"
    # do any settings of variables here for uppercase then shift
    shift
    ;;
  *) usage
    ;;
  esac
done
```

首先检查脚本是否有参数，如果没有，打印 usage语句，如果有需要处理的参数，使用case语句捕获每一个传送过来的选项。当处理完此选项后，使用 shift命令搜集命令行中下一选项，如果未发现匹配选项，打印usage语句。

当向脚本传递两个无效参数时，输出如下：

```
$ tr_case -u -l -k
-u option specified
-l option specified
usage:tr_case -[l|u] file [files]
```

下一步就是要用case语句处理选项后传递过来的文件名。为此需改动 case语句。case语句中捕获任意模式*应该为-*以允许传递无效选项，例如 -p或-q。

China-pub.com
下载

*模式也匹配传递过来的所有文件名，以便用 for循环处理每一个文件，这里也将使用 -f选项检测文件是否存在。

改动后的case语句如下：

```
case
...
-*) usage
  ;;
*)  if [ -f  $1 ]; then
     FILES=$FILES" " $1   # assign the filenames to a variable
    else
      echo "`basename $0` cannot find the file $1"
    fi
  shift  # get next one !
  ;;
esac
```

还需要指定与选项（-l，-u）相关的变量设置。这些变量是：

TRCASE　保存转换类型（大写或小写）。

EXT　所有文件转换后，大写文件名为.UC，小写为.LC，不保存初始文件状态。

OPT　如果给出此选项，设其为yes，否则为no。如果没有给出此选项，捕获此信息并反馈出来。

其他部分脚本用于实际转换处理，这里即 tr命令。tr命令放在case语句for循环中读取文件名进行处理的脚本末尾部分。

以下为完整脚本：

```
$ pg tr_case
!/bin/sh
# tr_case
# convert files to either upper or lower case
FILES=""
TRCASE=""
EXT=""
OPT=no

# gets called when a conversion fails
error_msg()
{
_FILENAME=$1
echo "`basename $0`: Error the conversion failed on $_FILENAME"
}

if [ $# -eq 0 ]
then
  echo "For more info try `basename $0` --help"
  exit 1
fi
while [ $# -gt 0 ]
do
  case $1 in
  # set the variables based on what option was used
  -u) TRCASE=upper
    EXT=".UC"
```

China-pub.com 下载

```
  OPT=yes
  shift
  ;;
-l) TRCASE=lower
  EXT=".LC"
  OPT=yes
  shift
  ;;
-help) echo "convert a file(s) to uppercase from lowercase"
        echo "convert a file(s) from lowercase to uppercase"
        echo "will convert all characters according to the"
        echo " specified command option."
        echo " Where option is"
        echo " -l Convert to lowercase"
        echo " -u Convert to uppercase"
        echo " The original file(s) is not touched. A new file(s)"
        echo "will be created with either a .UC or .LC extension"
        echo "usage: $0 -[l|u] file [file..]"
  exit 0
  ;;
-*) echo "usage: `basename $0` -[l|u] file [file..]"
  exit 1
  ;;

*) # collect the files to process
  if [ -f $1 ]
     then
       # add the filenames to a variable list
       FILES=$FILES" "$1
     else
       echo "`basename $0`: Error cannot find the file $1"
     fi
     shift
     ;;
   esac

done
# no options given ... help the user
if [ "$OPT" = "no" ]
then
  echo "`basename $0`:Error you need to specify an option. No action taken"
  echo " try `basename $0` --help"
  exit 1
fi

# now read in all the file(s)
# use the variable LOOP, I just love the word LOOP
for LOOP in $FILES
do
  case $TRCASE in
  lower) cat $LOOP|tr "[a-z]" "[A-Z]" >$LOOP$EXT
    if [ $? != 0 ]
    then
      error_msg $LOOP
```

China-pub.com 下载


```
      else
        echo "Converted file called $LOOP$EXT"
      fi
      ;;
    upper) cat $LOOP|tr "[A-Z]" "[a-z]" >$LOOP$EXT
      if [ $? != 0 ]
      then
        error_msg $LOOP
      else
        echo "Converted file called $LOOP$EXT"
      fi
      ;;
    esac
  done
```

执行上述脚本，给出不同选项，得结果如下：

转换一个不存在的文件：

```
$ tr_case -k cursor
usage: shiftl -[l|u] file [file..]
```

传递不正确选项：

```
$ tr_case cursor
tr_case:Error you need to specify an option. No action taken
 try tr_case -help
```

只键入文件名，希望脚本提示更多帮助信息：

```
$ tr_case
For more info try tr_case -help
```

输入两个有效文件及第三个无效文件：

```
$ tr_case -l cursor sd ascii
tr_case: Error cannot find the file sd
Converted file called cursor.LC
Converted file called ascii.LC
```

使用上述脚本可以将许多文件转换为同样的格式。编写一段脚本，使其控制不同的命令行选项，这种方式编程量很大，是一件令人头疼的事。

假定要写一段脚本，要求控制以下各种不同的命令行选项：

命令-l -c 23 -文件1文件2

shift命令显得力不从心，这就需要用到getopts命令。

20.2 getopts

getopts可以编写脚本，使控制多个命令行参数更加容易。 getopts用于形成命令行处理标准形式。原则上讲，脚本应具有确认带有多个选项的命令文件标准格式的能力。

20.2.1 getopts脚本实例

通过例子可以更好地理解getopts。以下getopts脚本接受下列选项或参数。

- a 设置变量ALL为true。
- h 设置变量HELP为true。

China-pub.com 下载

• f 设置变量FILE为true。
• v 设置变量VERBOSE为true。

对于所有变量设置，一般总假定其初始状态为 false：

```
$ pg getopt1
!/bin/sh
#getopt1

# set the vars
ALL=false
HELP=false
FILE=false
VERBOSE=false

while getopts ahfgv OPTION
do
  case $OPTION in
  a)ALL=true
    echo "ALL is $ALL"
    ;;
  h)HELP=true
    echo "HELP is $HELP"
    ;;
  f)FILE=true
    echo "FILE is $FILE"
    ;;
  v)VERBOSE=true
    echo "VERBOSE is $VERBOSE"
    ;;
  esac
done
```

getopts一般格式为：

```
getopts  option_string variable
```

在上述例子中使用脚本：

```
while getopts ahfgv OPTION
```

可以看出while循环用于读取命令行，option_string为指定的5个选项（-a，-h，-f，-g，-v），脚本中variable为OPTION。注意这里并没有用连字符指定每一单个选项。

运行上述脚本，给出几个有效和无效的选项，结果为：

```
$ getopt1 -a -h
ALL is true
HELP is true
$ getopt1 -ah
ALL is true
HELP is true

$ getopt1 -a -h -p
ALL is true
HELP is true
./getopt1: illegal option -- p
```

可以看出不同选项的结合方式。

20.2.2　getopts使用方式

getopts读取option_string，获知脚本中使用了有效选项。

getopts查看所有以连字符开头的参数，将其视为选项，如果输入选项，将把这与option_string对比，如果匹配发现，变量设置为 OPTION，如果未发现匹配字符，变量能够设置为？。重复此处理过程直到选项输入完毕。

getopts接收完所有参数后，返回非零状态，意即参数传递成功，变量 OPTION保存最后处理参数，一会儿就可以看出处理过程中这样做的好处。

20.2.3　使用getopts指定变量取值

有时有必要在脚本中指定命令行选项取值。 getopts为此提供了一种方式，即在option_string中将一个冒号放在选项后。例如：

```
getopts ahfvc: OPTION
```

上面一行脚本指出，选项 a、h、f、v可以不加实际值进行传递，而选项 c必须取值。使用选项取值时，必须使用变量 OPTARG保存该值。如果试图不取值传递此选项，会返回一个错误信息。错误信息提示并不明确，因此可以用自己的反馈信息屏蔽它，方法如下：

将冒号放在option_string开始部分。

```
while getopts :ahfgvc:  OPTION
```

在case语句里使用？创建一可用语句捕获错误。

```
case
...
...
\?)   # usage statement
  echo "`basename $0` -[a h f v] -[c value] file"
  ;;
esac
```

改动后getopts脚本如下：

```
$ pg getopt1
#!/bin/sh
#getopt1

# set the vars
ALL=false
HELP=false
FILE=false
VERBOSE=false
COPIES=0    # the value for the -c option is set to zero

while getopts :ahfgvc: OPTION
do
  case $OPTION in
  a)ALL=true
    echo "ALL is $ALL"
    ;;
  h)HELP=true
    echo "HELP is $HELP"
    ;;
```

```
   f)FILE=true
     echo "FILE is $FILE"
     ;;
   v)VERBOSE=true
     echo "VERBOSE is $VERBOSE"
     ;;
   c) COPIES=$OPTARG
     echo "COPIES is $COPIES"
   \?)    # usage statement
     echo "`basename $0` -[a h f v] -[c value] file" >&2
     ;;
   esac
 done
```

运行上述脚本，选项-c不赋值，将返回错误，但显示的是脚本语句中的反馈信息：

```
$ getopt1 -ah -c
ALL is true
HELP is true
getopt1 -[a h f v] -[c value] file
```

现在，输入所有合法选项：

```
$ getopt1 -ah -c 3
ALL is true
HELP is true
COPIES is 3
```

20.2.4 访问取值方式

getopts的一种功能是运行后台脚本。这样可以使用户加入选项，指定不同的磁带设备以备份数据。使用getopts实现此任务的基本框架如下：

```
$ pg backups
#!/bin/sh
# backups
QUITE=n
DEVICE=awa
LOGFILE=/tmp/logbackup
usage()
{
echo "Usage: `basename $0` -d [device] -l [logfile] -q"
exit 1
}
if [ $# = 0 ]
then
  usage
fi

while getopts :qd:l: OPTION
do
  case $OPTION in
  q) QUIET=y
     LOGFILE="/tmp/backup.log"
     ;;
  d) DEVICE=$OPTARG
     ;;
  l) LOGFILE=$OPTARG
```

China-pub.com
下载


```
        ;;
      \?) usage
        ;;
      esac
    done
    echo "you chose the following options..I can now process these"
    echo "Quite= $QUITE $DEVICE $LOGFILE"
```

上述脚本中如果指定选项d，则需为其赋值。该值为磁带设备路径。用户也可以指定是否备份输出到登录文件中的内容。运行上述脚本，指定下列输入：

```
$ backups -d/dev/rmt0 -q
you chose the following options..I can now process these
Quite= y /dev/rmt0 /tmp/backup.log
```

getopts检查完之后，变量OPTARG取值可用来进行任何正常的处理过程。当然，如果输入选项，怎样进行进一步处理及使该选项有有效值，完全取决于用户。

以上是使用getopts对命令行参数处理的基本框架。

实际处理文件时，使用for循环，就像在tr-case脚本中使用shift命令过滤所有选项一样。

使用getopts与使用shift方法比较起来，会减少大量的编程工作。

20.2.5 使用getopts处理文件转换

现在用所学知识将tr-case脚本转换为getopts版本。命令行选项getopts方法与shift方法的唯一区别是一个VERBOSE选项。

变量VERBOSE缺省取值为no，但选择了命令行选项后，case语句将捕获它，并将其设为yes，反馈的命令是一个简单的if语句。

```
if [ "VERBOSE" = "on" ]; then
  echo "doing upper on $LOOP..newfile called $LOOP$EXT"
fi
```

如果正在使用其他系统命令包，它总是反馈用户动作，只需简单地将包含错误的输出重定向到/dev/null中即可。如：

```
命令 >/dev/null 2 >&1
```

缺省时VERBOSE关闭（即不显示），使用-v选项可将其打开。例如要用 VERBOSE将myfile文件系列转换为小写，方法如下：

```
tr-case -l -v myfile1 myfile2 ...
```

或者

```
tr-case -v -l myfile1 myfile2 ...
```

可能首先注意的是使用 getopts后脚本的缩减效果。这里用于文件处理的脚本与 shift版本相同。

脚本如下：

```
$ pg tr_case2
#!/bin/sh
#tr_case2
# convert case, using getopts
EXT=""
TRCASE=""
FLAG=""
```

China-pub.com 下载

```
OPT="no"
VERBOSE="off"


while getopts :luv OPTION
do
  case $OPTION in
  l) TRCASE="lower"
    EXT=".LC"
    OPT=yes
    ;;
  u) TRCASE="upper"
    EXT=".UC"
    OPT=yes
    ;;
  v) VERBOSE=on
    ;;
  \?) echo "usage: `basename $0`: -[l|u] --v file[s]"
    exit 1 ;;
  esac
done
# next argument down only please
shift `expr $OPTIND - 1`
# are there any arguments passed ???
if [ "$#" = "0" ] || [ "$OPT" = "no" ]
then
  echo "usage: `basename $0`: -[l|u] -v file[s]" >&2
  exit 1
fi
for LOOP in "$@"
do
  if [ ! -f $LOOP ]
  then
    echo "`basename $0`: Error cannot find file $LOOP" >&2
    exit 1
  fi
  echo $TRCASE $LOOP
  case $TRCASE in
  lower) if [ "VERBOSE" = "on" ]; then
           echo "doing..lower on $LOOP..newfile called $LOOP$EXT"
         fi
    cat $LOOP | tr "[a-z]" "[A-Z]" >$LOOP$EXT
    ;;
  upper) if [ "VERBOSE" = "on" ]; then
           echo "doing upper on $LOOP..newfile called $LOOP$EXT"
         fi
    cat $LOOP | tr "[A-Z]" "[a-z]" >$LOOP$EXT
    ;;
  esac
done
```

在脚本中指定命令行选项时，最好使其命名规则与 UNIX或LINUX一致。下面是一些选项及其含义的列表。

China-pub.com
下载

选　项	含　义
-a	扩展
-c	计数、拷贝
-d	目录、设备
-e	执行
-f	文件名、强制
-h	帮助
-i	忽略状态
-l	注册文件
-o	完整输出
-q	退出
-p	路径
-v	显示方式或版本

20.3 小结

正确控制命令行选项会使脚本更加专业化，对于用户来说会使之看起来像一个系统命令。本章讲到了控制命令行选项的两种方法， shift和getopts。使用getopts检测脚本的数量远远小于使用shift方法检测脚本的数量。

shift也克服了脚本参数 $1..$9的限制。使用shift命令，脚本可以很容易偏移至所有调用参数，因此脚本可以做进一步处理。

China-pub.com 下载

第21章 创建屏幕输出

用户可以使用shell脚本创建交互性的、专业性强的屏幕输出。要实现这一点，系统上需要一个彩色监视器和tput命令。

本章内容有：

• tput命令。

• 使用转义序列和产生控制码。

• 使用颜色。

作者写这本书时，遇到了tput命令的三种不同变形。至今为止最好的是GNU tput，如果没有这个版本，首先下载它并安装在你的系统里。 tput使用文件/etc/terminfo或/etc/termcap，这样就可以在脚本中使用终端支持的大部分命令了。

虽然tput不识别颜色设置，但是可以使用控制字符实现这一点。

21.1 tput

在使用tput前，需要在脚本或命令行中使用tput命令初始化终端。

```
$ tput init
```

tput产生三种不同的输出：字符型、数字型和布尔型（真 /假）。以下分别介绍其使用功能。

21.1.1 字符串输出

下面是大部分常用字符串：

名　字	含　义
bel	警铃
blink	闪烁模式
bold	粗体
civis	隐藏光标
clear	清屏
cnorm	不隐藏光标
cup	移动光标到屏幕位置（x，y）
el	清除到行尾
ell	清除到行首
smso	启动突出模式
rmso	停止突出模式
smul	开始下划线模式
rmul	结束下划线模式
sc	保存当前光标位置
rc	恢复光标到最后保存位置
sgr0	正常屏幕
rev	逆转视图

China-pub.com 下载

21.1.2 数字输出

以下是大部分常用数字输出。

名 字	含 义
cols	列数目
it	tab设置宽度
lines	屏幕行数

21.1.3 布尔输出

在tput中只有两种布尔操作符。

名 字	含 义
chts	光标不可见
hs	具有状态行

21.2 tput用法

上面讲过了可能用到的tput的大多数常用名。现在学习在一段脚本中使用 tput。

21.2.1 设置tput命令

可以取得所有tput名字输出，将其保存为更有意义的变量名。格式如下：

```
variable_name='tput name'
```

21.2.2 使用布尔输出

可以在if语句中使用布尔型tput输出。

```
STATUS_LINE='tput hs'
if $STATUS_LINE; then
  echo "your terminal has a status line"
else
  echo "your terminal has NO status line"
fi
```

21.2.3 在脚本中使用tput

以下脚本设置tput bel和cl为更有意义的变量名。

```
$ pg tput1
#!/bin/sh
BELL=`tput bel`
CLEAR=`tput cl`

echo $BELL
echo $CLEAR
```

下面脚本改变两个视图属性，并将光标关闭和打开。

```
$ pg tput2
#!/bin/sh
```

China-pub.com 下载

```
BOLD=`tput bold`
REV=`tput rev`
NORMAL=`tput sgr0`
CURSOR_OFF=`tput civis`
CURSOR_ON=`tput cnorm`
tput init

#turn cursor off, highlight text, reverse some text, cursor on
echo $CURSOR_OFF
echo "${BOLD} WELCOME TO THE PIZZA PLACE${NORMAL}"
echo -e "\n${REV} WE ARE OPEN 7 DAYS A WEEK${NORMAL}"
echo $CURSOR_ON
```

21.2.4 产生转义序列

注意，如果正在使用一个仿真器，要使光标不可见，这个操作可能会有问题。这是因为：

1) 一些仿真器并不捕获使光标不可见的控制字符。必须要求正在使用的软件仿真的制作者修改源脚本以关闭光标。

2) tput civis命令的一些旧版本工作不正常。

关闭光标的控制字符是？25l（这是字母l），将之打开是？25h。

所有控制字符均以一个转义序列开始。通常转义键后紧跟字符 [。然后实际序列打开或关闭某终端属性。

可以使用两种不同的方法产生转义序列。下面的列表依据用户系统列出两种方法。第三种方法对于UNIX和LINUX支持的变量均有效，因为控制序列嵌在echo语句中。本书将使用这种方法。

要发送一转义序列以关闭光标：

LINUX/BSD	`echo -e "\033[?25l"`
System V	`echo "\033[?25l"`
Generic method	`echo "<CTRL-V><ESCAPE>[?25l"`

\033为转义键取值，\通知echo命令接下来是一个八进制值。例如要反馈一个 @字符，键入：

echo “@”或者**echo-e“\100”**

对于系统v，使用

echo “\100”

结果是一样的。

命令clear表示清屏并发送光标到屏幕左上角，此位置一般也称为 home。在一个VT终端范围实现此功能所需序列为ESCIJ，可以使用echo语句发送这一序列。

System V	`echo "\033[2J"`
LINUX/BSD	`echo -e "\033[2J"`

对于嵌入在文本中的任何控制字符，不要试图剪切和粘贴，因为这样会失去其特殊含义。例如，要插入控制字符，打开光标，方法如下：

```
echo'<CTRL-V> hit the<ESCAPE> key then [?25h'
```

即先击<CTRL-V>,再击退格键，确保这不是一个仿真器。然后加入一小段脚本将之

打开和关闭。可以将这段脚本编成一个函数或者在后面几页找一下这段脚本。

```
$ pg cursor
#!/bin/sh

# cursor  on|off
# turns the cursor on or off for the vt100, 200, 220, meth220
# note : will work on normal tty connec.. if'ie on some win emulations
# check TERM env for your type !
_OPT=$1
if [ $# -ne 1 ]; then
  echo "Usage:`basename $0` cursor [on|off]"
  exit 1
fi

case "$_OPT" in
on|ON|On)
  # turn it on (cursor)
  ON=`echo ^[[?25h`
  echo $ON
  ;;
off|OFF|Off)
  # turn it off (cursor)
  OFF=`echo ^[[?25l`
  echo $OFF
  ;;
;;
*) echo "Usage: cursor on|off"
  exit 1
  ;;
esac
```

21.2.5 光标位置

可以用tput将光标放在屏幕任意位置。格式为：

```
cup r c
```

r为从上至下屏幕行数， c为穿过屏幕列数。

最好将之编成函数，这样就可以把行和列的值传递给它。

```
xy()
{
#_R= row, _C=column
_R=$1
_C=$2
tput cup $_R $_C
}

clear
xy 1 5
echo -n "Enter your name :"
read NAME
xy 2 5
echo -n "Enter your age  :"
read AGE
```

当然再传递一个字符串给它也很合适。以下是稍加改动后的函数脚本。

China-pub.com 下载

```
xy()
{
#_R= row, _C=column
_R=$1
_C=$2
_TEXT=$3
tput cup $_R $_C
echo -n  $_TEXT
}
```

这可以像下面这样调用：

```
xy 5 10 "Enter your password :"
read CODE
```

21.2.6 在屏幕中心位置显示文本

在屏幕中心位置显示文本不是很麻烦。首先从 tput中得到列数，然后算出所提供的字符串长度，从tput列数中减去该值，结果再除以2，所得结果可用于显示的字符串的行数。

以下脚本段实现此功能。只需稍加改动即可从文件中读取各行并在屏幕中间位置显示文本。

输入一些字符，点击回车键，文本将显示在屏幕中间第 10行。

```
echo -n "input string :"
read STR
# quick way of getting length of string
LEN=`echo $STR | wc -c`
COLS=`tput cols`
NEW_COL=`expr \($COLS - $LEN \) / 2`
xy 10 $NEW_COL
echo $STR
```

将上述脚本编成函数，并带有两个参数：文本和行数，这样调用更加灵活，函数如下：

```
centertxt()
{
_ROW=$1
_STR=$2
# quick way of getting length of string
LEN=`echo $_STR | wc -c`
COLS=`tput cols`
_NEW_COL=`expr \($COLS -- $LEN \) / 2`
xy $_ROW $_NEW_COL
echo  $_STR
}
```

可使用下述格式调用上述函数：

```
centertxt 15“THE MAIN EVENT”
```

或者用字符串作参数 ：

```
centertxt 15 $1
```

21.2.7 查找终端属性

下面脚本使用tput访问terminfo，显示前面提到过的tput命令下的一些终端转义码。

```
$ pg termput
#!/bin/sh
# termput
```

```
#init tput for your terminal
tput init

clear

echo " tput <> terminfo"
infocmp -1 $TERM | while read LINE
do
  case $LINE in
  bel*)   echo "$LINE: sound the bell" ;;
  blink*) echo "$LINE: begin blinking mode" ;;
  bold*)  echo "$LINE: make it bold" ;;
  el*)    echo "$LINE: clear to end of line" ;;
  civis*) echo "$LINE: turn cursor off" ;;
  cnorm*) echo "$LINE: turn cursor on " ;;
  clear*) echo "$LINE: clear the screen ";;
  kcuu1*) echo "$LINE: up arrow ";;
  kcub1*) echo "$LINE: left arrow ";;
  kcuf1*) echo "$LINE: right arrow ";;
  kcud1*) echo "$LINE: down arrow ";;
  esac
done
```

命令infocmp从terminfo数据库中抽取终端信息，如果要查看终端定义文件的完整列表，可使用命令：

```
$ infocmp $TERM
```

以下是termput脚本的终端输出：

```
$ termput
tput <> terminfo
bel=^G,: sound the bell
blink=E[5m,: begin blinking mode
bold=E[1m,: make it bold
civis=E[?25l,: turn cursor off
clear=E[HE[J,: clear the screen
cnorm=E[?25h,: turn cursor on
el=E[K,: clear to end of line
el1=E[1K,: clear to end of line
kcub1=E[D,: left arrow
kcud1=E[B,: down arrow
kcuf1=E[C,: right arrow
kcuu1=E[A,: up arrow
```

21.2.8 在脚本中使用功能键

使用cat命令可以查看发送的任意特殊键控制序列（ F1，上箭头等），键入cat -v，然后按任意控制键，回车，在下一行就可以知道终端发送了什么功能键。结束后按 <Ctrl-c>退出。

下面的例子运行cat命令，输入键为F1（^[OP]），F2([OQ])，上箭头[^[[A]。

```
$ cat -v
^[OP^[OQ^[[A
<CTRL-C>
```

有了这些信息，就可以在脚本中插入这些字符作为用户选择的另外一些方法。

下面脚本识别F1、F2和箭头键，取值可能不同，因此要先用 cat命令查看用户终端控制键发送的取值。

China-pub.com 下载

```
$ pg control_keys
#!/bin/sh
# control_keys
# to insert use '<CTRL-V><ESCAPE>sequence'
uparrowkey='^[[A'
downarrowkey='^[[B'
leftarrowkey='^[[D'
rightarrowkey='^[[C'
f1key='^[OP'
f2key='^[OQ'

echo -n " Press a control key then hit return"
read KEY

case $KEY in
$uparrowkey) echo "UP Arrow"
  ;;
$downarrowkey) echo "DOWN arrow"
  ;;
$leftarrowkey) echo "LEFT arrow"
  ;;
$rightarrowkey) echo "RIGHT arrow"
  ;;
$f1key) echo "F1 key"
  ;;
$f2key) echo "F2 key"
  ;;
*) echo "unknown key $key"
  ;;
esac
```

21.2.9 使用颜色

对域使用颜色可以使数据输入屏幕看起来更加专业。下面将使用的颜色是 ANSI标准颜色，并不是所有颜色都适合于所有系统。下面列出了大部分常用颜色。

1. 前景色：

数 字	颜 色	数 字	颜 色
30	黑色	34	蓝色
31	红色	35	紫色
32	绿色	36	青色
33	黄（或棕）色	37	白（或灰）色

2. 背景色：

数 字	颜 色	数 字	颜 色
40	黑色	44	青色
41	红色	45	蓝色
42	绿色	46	青色
43	黄（或棕）色	47	白（或灰）色

显示前景或背景颜色格式为：

```
<ESCAPE> [background_number;foreground_number m
```

21.2.10 产生颜色

产生颜色需要在echo语句中嵌入控制字符。这种方法适用于带有彩色终端的任何系统。与在控制字符里一样，可以在echo语句里使用转义序列产生颜色。

要产生一个黑色背景加绿色前景色：

LINUX/BSD	`echo -e "\033[40;32m"`
System V	`echo "\033[40;32m"`
Generic method	`echo "<CTRL-V><ESCAPE>[40;32m"`

一般方法是先击<Ctrl-v>，然后是<ESCAPE>键，接着是[40；32 m。本书使用这种方法。

可能发现将颜色设置与echo语句放在一个case语句里，然后将之编成一个函数，这样做最好。下面是作者编写的颜色函数。

```
colour()
{
# format is background;foregroundm
case $1 in
black_green)
  echo '^[[40;32m'
  ;;
black_yellow)
  echo '^[[40;33m'
  ;;
black_white)
  echo '^[[40;37m'
  ;;
black_cyan)
  echo '^[[40;36m'
  ;;
red_yellow)
  echo '^[[41;32m'
  ;;
black_blue)
  echo '^[[40;34m'
  ;;
esac
}
```

要调用颜色red-yellow（红色背景，黄色前景），方法如下：

```
colour red-yellow
```

在脚本中可以这样使用颜色：

```
colour what_ever
echo something
# now change to a different colour
colour what_ever
echo something
```

作者终端的缺省屏幕颜色是黑色和白色。但是如果要用黑色背景加绿色前景，可插入一个echo语句，同时将之放入用户.profile文件中。

图21-2显示加入颜色设置后的基本输出屏幕。这种颜色看起来更加吸引人

China-pub.com 下载

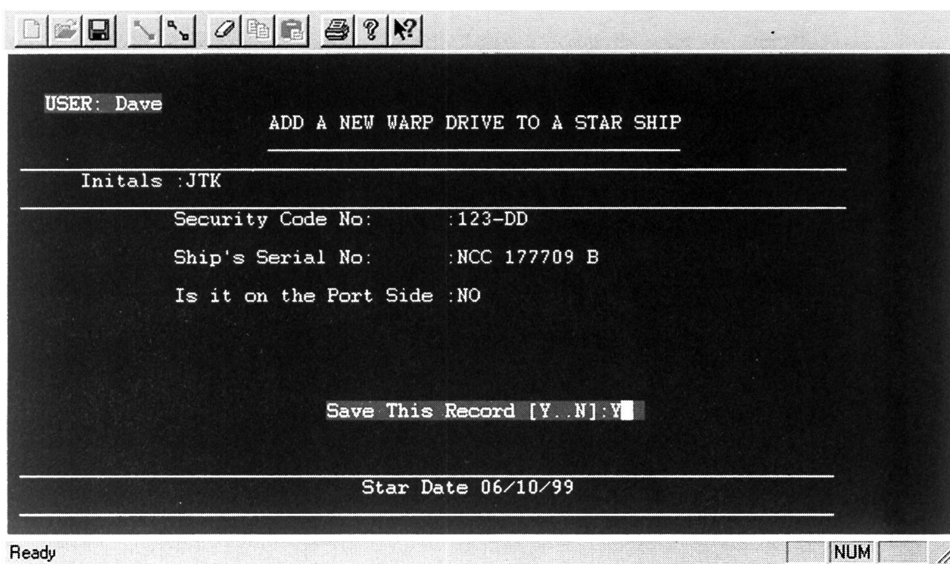


图21-1 下述脚本使用颜色和tput命令所得的屏幕概貌

下面是显示图21-1屏幕的脚本：

```
$ pg colour_scr
#!/bin/sh
# colour_scr
tput init
MYDATE=`date +%D`
colour()
{
# format is background;foregroundm
case $1 in
black_green)
  echo '^[[40;32m'
  ;;
black_yellow)
  echo '^[[40;33m'
  ;;
black_white)
  echo '^[[40;37m'
  ;;
black_cyan)
  echo '^[[40;36m'
  ;;
black_red)
  echo '^[[40;31m'
  ;;
esac
}

xy()
# xy
# to call: xy row, column, "text"
# goto xy screen co-ordinates
{
```

China-pub.com
下载


```
#_R= row, _C=column
_R=$1
_C=$2
_TEXT=$3
tput cup $_R $_C
echo -n $_TEXT
}

center()
{
# center
# centers a string of text across screen
# to call: center "string" row_number
_STR=$1
_ROW=$2
# crude way of getting length of string
LEN=`echo $_STR | wc -c`

COLS=`tput cols`
HOLD_COL=`expr $COLS - $LEN`
NEW_COL=`expr $HOLD_COL / 2`
tput cup $_ROW $NEW_COL
echo -n $_STR
}

tput clear
colour red_yellow
xy 2 3 "USER: $LOGNAME"
colour black_cyan
center "ADD A NEW WARP DRIVE TO A STAR SHIP" 3
echo -e "\f\f"
center "___________________________________" 4

colour black_yellow
xy 5 1  "________________________________________________________________"
xy 7 1  "________________________________________________________________"
xy 21 1 "________________________________________________________________"
center "Star Date $MYDATE " 22
xy 23 1 "________________________________________________________________"

colour black_green
xy 6 6 "Initials :"
read INIT
xy 8 14
echo -n "Security Code No:      :"
read CODE
xy 10 14
echo -n "Ship's Serial No:      :"
read SERIAL
xy 12 14
echo -n "Is it on the Port Side :"
read PORT

colour red_yellow
center " Save This Record [Y..N]: " 18
read ans
```

China-pub.com 下载

```
#reset to normal
colour black_white
```

如你所见，这个脚本没有经过验证，这样也行，因为这里脚本的目标只是显示怎样为屏幕上色。

21.2.11 创建精致菜单

在讲述while循环时曾经创建过菜单，现在增加菜单脚本，菜单将具有下列选项：

1 : ADD A RECORD
2 : VIEW A RECORD
3 : PAGE ALL RECORDS
4 : CHANGE A RECORD
5 : DELETE A RECORD
P : PRINT ALL RECORDS
H : Help screen
Q : Exit Menu

本脚本使用read_char函数，使用户在选择菜单选项时不必敲入回车键。 trap命令（本书后面提到）用于忽略信号2、3和15，这样将防止用户试图跳出菜单。

此菜单还有一些控制访问形式。授权用户可以修改和删除记录，其余用户只能增加，查看或打印记录。带有访问级别的有效用户列表保存在文件 priv.user中。

用户运行菜单时，如果菜单名在文件中不存在，将被告之不能运行此应用并且退出。

只出于显示目的，系统命令就替换了实际的选项执行操作。执行时我们会发现用户 root，dave和matty不能修改数据库文件，而peter和louise可以。

```
$ pg priv.user
# prov.user access file for apps menu
# edit this at your own risk !!!!
# format is USER AMEND/DELETE records
# example root yes means yes root can amend or delete recs
#    "     dave no  means no dave cannot amend or delete recs
root no
dave no
peter yes
louise yes
matty no
```

要检查用户权限，首先需要读入列表文件，忽略注释行，将其他行重定向到一个临时文件中。

```
user_level()
{
while read LINE
do
  case $LINE in
  \#*);;
  *) echo $LINE >>$HOLD1
    ;;
  esac
done < $USER_LEVELS

FOUND=false
```

China-pub.com
下载


```
while read MENU_USER PRIV
do
  if [ "$MENU_USER" = "$USER" ];
  then
    FOUND=true
    case $PRIV in
    yes|YES)
      return 0
      ;;
    no|NO)
      return 1
      ;;
    esac
   else
     continue
   fi
 done <$HOLD1
   if [ "$FOUND" = "false" ]; then
   echo "Sorry $USER you have not been authorised to use this menu"
   exit 1
   fi
 }
```

下一步是读取新形成的格式化文件，变量FOUND首先设置为假，临时文件保存名字和权限级别。分别用用户名和权限级别设置为一个变量，然后执行测试文件中名字是否匹配USER。USER取值是从脚本开始时whoami命令中获得的。如果不匹配，则寻找其他用户，使用命令continue循环进入下一步。

处理过程直至所有用户名读取和匹配完毕。如果整个文件中均未找到匹配用户名，脚本末尾的test语句捕获权限级别，对一般访问级别为1，对高级访问权限返回0。

当用户选择修改或删除记录时，基于上述函数的返回值进行了一项测试。这个例子中passwd文件被分类或列出一个目录清单。

```
if user_level; then
  sort /etc/passwd
else
  restrict
fi
```

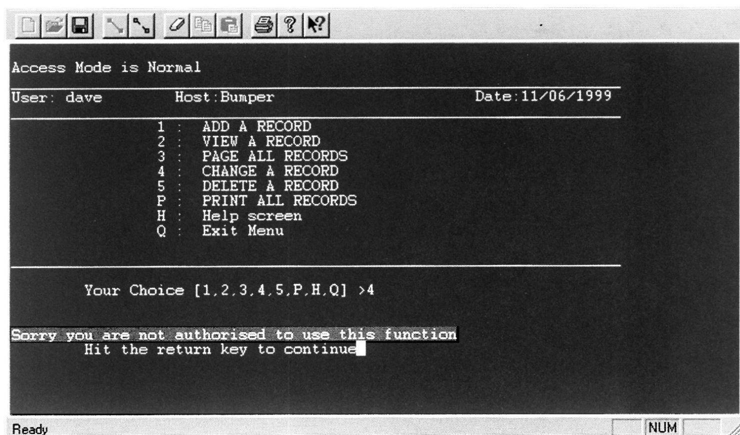


图21-2　带有访问限制的菜单屏幕概貌

China-pub.com 下载

Restrict是一个只打印违规操作提示符的函数。

上述测试在一个循环里即可完成，但考虑到脚本清晰性，使用了两个文件的方法，这样调试起来会更容易。

图21-2显示用户dave试图修改记录，但只具有一般权限，因此被提示权限不够。

用户可以选择q或Q退出菜单。退出时调用一个清屏函数。这样做的好处在于可以随意增加要运行的命令，同时也增加脚本的可读性。

脚本如下：

```
$ pg menu2
#!/bin/sh
# menu2
# MAIN MENU SCRIPT
# ignore CTRL-C and QUIT interrupts
trap "" 2 3 15
MYDATE=`date +%d/%m/%Y`
THIS_HOST=`hostname -s`

USER=`whoami`

# user level file
USER_LEVELS=priv.user

# hold file
HOLD1=hold1.$$

# colour function
colour()
{
# format is background;foregroundm
case $1 in
black_green)
  echo '^[[40;32m'
  ;;
black_yellow)
  echo '^[[40;33m'
  ;;
black_white)
  echo '^[[40;37m'
  ;;
black_cyan)
  echo '^[[40;36m'
  ;;
red_yellow)
  echo '^[[41;33m'
  ;;
esac
}

# just read a single key please
get_char()
{
```

```
# get_char
# save current stty settings
SAVEDSTTY=`stty -g`
  stty cbreak
  dd if=/dev/tty bs=1 count=1 2> /dev/null
  stty -cbreak
# restore stty
stty $SAVEDSTTY
}

# turn the cursor on or off
cursor()
{
# cursor
#turn cursor on/off

_OPT=$1
  case $_OPT in
  on) echo '^[[?25h'
    ;;
  off) echo '^[[?25l'
    ;;
  *) return 1
    ;;
  esac
}

# check what privilege level the user has
restrict()
{
colour red_yellow
echo -e -n "\n\n\007Sorry you are not authorised to use this function"
colour black_green
}
user_level()
{
# user_level
# read in the priv.user file
while read LINE
do
  case $LINE in
  # ignore comments
  \#*);;
  *) echo $LINE >>$HOLD1
    ;;
  esac
done < $USER_LEVELS

FOUND=false
while read MENU_USER PRIV
do
  if [ "$MENU_USER" = "$USER" ];
  then
    FOUND=true
```

China-pub.com 下载

```
      case $PRIV in
      yes|YES)
        return 0
        ;;
      no|NO)
        return 1
        ;;
      esac
    else
    # no match found read next record
      continue
    fi
done <$HOLD1
if [ "$FOUND" = "false" ]; then
  echo "Sorry $USER you have not been authorised to use this menu"
  exit 1
fi
}

# called when user selects quit
my_exit()
{
# my_exit
# called when user selects quit!
colour black_white
  cursor on
  rm *.$$
  exit 0
}
tput init
# display their user levels on the screen
if user_level; then
  ACCESS="Access Mode is High"
else
  ACCESS="Access Mode is Normal"
fi

tput init
while :
do
tput clear
colour black_green
cat <<MAYDAY
$ACCESS
----------------------------------------------------------------------------------------
User: $USER          Host:$THIS_HOST                         Date:$MYDATE
----------------------------------------------------------------------------------------
                    1 :  ADD A RECORD
                    2 :  VIEW A RECORD
                    3 :  PAGE ALL RECORDS
                    4 :  CHANGE A RECORD
                    5 :  DELETE A RECORD
                    P :  PRINT ALL RECORDS
                    H :  Help screen
                    Q :  Exit Menu
----------------------------------------------------------------------------------------
```

China-pub.com
下载


```
MAYDAY
colour black_cyan
echo -e -n "\tYour Choice [1,2,3,4,5,P,H,Q] >"
œread CHOICE
CHOICE=`get_char`
  case $CHOICE in
  1) ls
    ;;
  2) vi
    ;;
  3) who
    ;;
  4) if user_level; then
       ls -l |wc
     else
       restrict
     fi
    ;;
  5) if user_level; then
       sort /etc/passwd
     else
       restrict
     fi
    ;;
  P|p) echo -e "\n\nPrinting records......."
    ;;
  H|h)
  tput clear
  cat <<MAYDAY
  This is the help screen, nothing here yet to help you!
  MAYDAY
    ;;
  Q|q) my_exit
    ;;
  *) echo -e "\t\007unknown user response"
    ;;
  esac
echo -e -n "\tHit the return key to continue"
read DUMMY
done
```

这种菜单可以在profile文件中用exec命令调用，用户不能够退出。它们将始终位于菜单或菜单子选项的应用里面。这对于只使用 UNIX或LINUX应用而不关心shell的用户来说是一种好方法。

21.3 小结

使用tput命令可以增强应用外观及脚本的控制。颜色设置可以增加应用的专业性。注意使用颜色不要太过火，这也许对你来说很好，但其他用户使用这段脚本时看到这种屏幕设置也许会感到厌烦。可以使用和读取控制字符来增加脚本的灵活性，特别是对用户击键输入操作更是如此。

China-pub.com 下载

第22章　创建屏幕输入

屏幕输入或数据输入是接受输入（这里指键盘）并验证其有效的能力。如果有效，接受它，如果无效，放弃该输入。

前面讲到了基于一些条件的测试函数，例如字符串长度、字符串是数字或字符型，这一章在脚本中将继续使用这些函数。

本章内容有：

• 验证有效输入。

• 增加、删除、修改和查看记录。

• 修改脚本的工作文件。

本章开始读起来可能有些累人，因此可以先大概看一下，再慢慢细看。验证有效性的代码量很大，这是因为为捕获所有错误，脚本必须测试几乎所有可能的错误。

现在在创建一个一般文件以修改系统地过程中逐步实现每一个任务：增加、删除、修改和查看记录。这里也将创建一个个人文件以修改系统。记录文件 DBFILE保存下列信息：

域	长　度	允许输入	描　述
职员号码	10	数字	雇员代码
名	20	字符	雇员名
姓	20	字符	雇员姓
部门	-	记帐	雇员所在部门
		IT	
		服务	
		销售	
		权利	

域间用冒号：分隔，例如：

```
<Staff number>:<First name>:<Second name>:<Department>
```

每一个任务即是一个完整脚本。脚本中一小部分复制于前面的两个例子。这样做是因为本章主要用于显示怎样用文件修改系统。刚开始编写脚本时，最令人头疼的事就是将修改的文件或数据库系统放在一起后的文档清理工作。

运行脚本应具有一些菜单选项，与任务或模块相连接或包含在文件里与菜单脚本相关的一系列函数相连接。每一段脚本均执行 trap命令，信号2、3和15被忽略。

22.1　增加记录

将记录加入文件，包含以下两个步骤：

1) 确认输入有效。

2) 将记录写入文件。

第一个任务就是将一些函数放在一起，这些函数测试域是字符型或数字型及域的长度限制，即数据输入有效性检验。有效性检验将用于增加数据和修改数据。幸运的是前面已经编

好这些函数，检测字符串及长度的函数脚本如下：

```
length_check()
{
# length_string
# $1=string, $2= length of string not to exceed this number
_STR=$1
_MAX=$2
_LENGTH=`echo $_STR |awk '{print length($0)}'`
if [ "$_LENGTH" -gt "$_MAX" ]; then
  return 1
else
  return 0
fi
}
```

检测字符串是否为数字型，函数脚本如下：

```
a_number()
#a _number
# $1= string
{
_NUM=$1
_NUM=`echo $1|awk '{if($0~/[^0-9]/) print "1"}'`
if [ "$_NUM" != "" ]
then
  return 1
else
  return 0
fi
}
```

检测字符串是否为字符型，函数脚本如下：

```
characters()
# characters
# $1 = string
{
_LETTERS_ONLY=$1
_LETTERS_ONLY=`echo $1|awk '{if($0~/[^a-zA-Z]/) print "1"}'`

if [ "$_LETTERS_ONLY" != "" ]
then
  return 1
else
  return 0
fi
```

当域读取完毕时，调用相应函数，测试其返回值。

这里也需要提示以保存屏幕信息直到键入某键删除这些信息，下列函数用到了read_a_char函数。

```
continue_promptYN()
{
# continue_prompt
echo -n "Hit any key to continue.."
DUMMY=`read_a_char`
}
```

China-pub.com 下载

当用户输入雇员代码后，要确保编号还没有用到，此域必须唯一。测试这一点有几种方法，这里使用grep。grep搜寻字符串_CODE中的雇员编号，如果awk返回空值，则不存在匹配编号，函数返回状态码0。函数如下（这里在grep中使用“ $_CODE\> ”抽取相应匹配，变量用双引号括起来，如果用单引号，则返回空值）：

```
check_duplicate()
{
# check_duplicate
# check for employee number duplicate
_CODE=$1
MATCH=`grep "$_CODE\>" $DBFILE`
echo $_CODE
if [ "$MATCH" = "" ]; then
  return 0  # no duplicate
else
  return 1 # duplicate found
fi
}
```

以下是检测雇员编号代码段，之后继续讲解其功能。

```
while :
do
echo -n "Employee Staff Number :"
read NUM
# check for input
if [ "$NUM" != "" ]; then
  if a_number $NUM; then
  # number OK
    NUM_PASS=0
  else
    NUM_PASS=1
  fi
  if length_check $NUM 10; then
  # length OK
    LEN_PASS=0
  else
    LEN_PASS=1
  fi
  # now check for duplicates...

  if check_duplicate $NUM; then
  # no duplicates
    DUPLICATE=0
  else
    DUPLICATE=1
    echo "Staff Number: There is already an employee with this number"
    continue_prompt
  fi
  # check all three variables now, they must all be true
  if [ "$LEN_PASS" = "0"  -a  "$NUM_PASS" = "0" -a "$DUPLICATE" = "0" ]
  then
    break
  else
    echo "Staff Number: Non-Numeric or Too Many Numbers In Field"
```

China-pub.com
下载


```
    continue_prompt
  fi
  else
    echo "Staff Number: No Input Detected, This Field Requires a Number"
    continue_prompt
  fi

  done
```

所有检测语句均在while循环中（实际上每一个数据输入域均在一单独的 while循环中），如果没有有效数据，提示返回初始读位置。

读完雇员编号，继续检测域中数据是否存在：

```
if ["$NUM"!=""]
```

如果域中没有输入数据。则不执行 if语句then部分。else部分在脚本结尾部分，用于显示下列信息：

```
Staff Number:No Input Detected,This Field Requires a Number
```

then部分包括对域输入数据的所有有效性检测。假定存在输入，调用 a_number函数，测试传输字符串是否为一数字，如果是，函数返回 0，如果不是，函数返回 1。基于此返回值，设置指针NUM_PASS为0，表示返回值正确（数字型），设置为1，表示返回失败（非数字型）。

然后调用函数length_check，参数为字符串及字符串包含字符最大数目。这里为 10，如果字符串长度小于10，则返回0，否则返回1。指针LEN_PASS设置为0，表示返回成功（长度不超过最大长度），设置为1，表示返回失败（长度超出最大长度）。

接下来检测是否有重复雇员编号。调用函数 check_duplicate，如果没有发现重复编号，设置指针DUPLICATE为0，最后测试三个指针变量均为 0（无错误），为此使用AND测试，如果条件同时成立，执行then部分语句。

如果测试通过，则存在有效域。这时处在 while循环中，因此需要用break命令跳出循环。

```
if [ "$LEN_PASS" = "0" -a "$NUM_PASS" = "0" -a "$DUPLICATE" = "0"
]; then
break
```

如果有效性测试任何一部分失败，即长度测试或类型测试之一不通过，返回错误信息并显示在屏幕底部。

```
Staff Number : Non_Numeric or Too Many Numbers In Field
```

验证第2和第3域有效性，处理过程一样。有效性验证这次在另一个循环中。这次调用characters函数，检验域是否只包含字符。下述脚本做名字有效性检验：

```
while :
do
echo -n  "Employee's First Name :"
read F_NAME
if [ "$F_NAME" != "" ]; then
  if characters $F_NAME; then
    F_NAME_PASS=0
  else
    F_NAME_PASS=1
  fi
  if length_check $F_NAME 20; then
```

China-pub.com 下载

```
    LEN_PASS=0
  else
    LEN_PASS=1
  fi
  if [ "$LEN_PASS" = "0"  -a  "$F_NAME_PASS" = "0" ]; then

    break
  else
    echo "Staff First Name: Non-Character or Too Many Characters In Field"
    continue_prompt
  fi
else
  echo "Staff First Name: No Input Detected, This Field Requires
        Characters"
  continue_prompt
fi
done
```

使用case语句检验部门域（列表见下面），因为公司只包含5个部门，部门域必须是其中之一。注意对每个部门有三个不同的匹配模式，可以由用户键入部门名称加以验证。如果找到匹配模式，用户跳出case语句，并显示有效部门列表。

```
while :
do
echo -n  "Company Department    :"
read DEPART
  case $DEPART in
  ACCOUNTS|Accounts|accounts) break;;
  SALES|Sales|sales)break;;
  IT|It|it) break;;
  CLAIMS|Claims|claims)break;;
  SERVICES|Services|services)break;;
  *) echo "Department: Accounts,Sales,IT,Claims,Services";;
  esac
done
```

当所有域的有效性验证完成后，将提示用户是否保存此记录，这里使用函数continue_promptYN，前面讲过这个函数，它处理Y或N响应，用户也可以点击回车键表示缺省回答。

如果用户选择N，进入if语句代码段并退出脚本。如果用户输入许多记录，然后在while循环中调用函数以增加记录，这种方法将不会返回菜单或增加记录后退出。

如果用户选择Y，保存记录。将记录加入一个文件的脚本是：

```
echo "$NUM:$F_NAME:$S_NAME:$DEPART">>$DBFILE
```

然后显示信息通知用户记录已存入文件。 sleep命令将脚本进程挂起1s，以使用户有足够的时间查看该信息。

域分隔符是冒号，文件以姓域分类，输出存入一临时工作文件，然后文件移入初始文件DBFILE。文件转移操作时，会测试其最后状态，如果存在问题，则通知用户该信息。

加入一个记录后，输出如下：

```
ADD A RECORD
        Employee Staff Number :23233
```

```
        Employee's First Name :Peter
        Employee's Surname    :Wills
        Company Department    :Accounts
Do You wish To Save This Record  [Y..N] [Y]:
saved
```

增加记录后，DBFILE文件内容如下：

```
$ pg DBFILE
32123:Liam:Croad:Claims
2399:Piers:Cross:Accounts
239192:John:Long:Accounts
98211:Simon:Penny:Services
99202:Julie:Sittle:IT
23736:Peter:Wills:Accounts
89232:Louise:Wilson:Accounts
91811:Andy:Wools:IT
```

以下是增加一个记录的完整脚本：

```
$ pg dbase_add
#!/bin/sh
# dbase_add
# add a record
#ignore signals
trap ""  2 3 15
# temp hold files
DBFILE=DBFILE
HOLD1=HOLD1.$$

read_a_char()
{
# read_a_char
# save the settings
SAVEDSTTY=`stty -g`
  stty cbreak
  dd if=/dev/tty bs=1 count=1 2> /dev/null
  stty -cbreak
stty $SAVEDSTTY
}

continue_promptYN()
#to call: continue_prompt "string to display" default_answer
{
# continue_prompt
_STR=$1
_DEFAULT=$2
# check we have the right params
if [ $# -lt 1 ]; then
  echo "continue_prompt: I need a string to display"
  return 1
fi
while :
do
  echo -n "$_STR [Y..N] [$_DEFAULT]:"
  read _ANS
```

China-pub.com 下载

```
# if user hits return set the default and determine the return value
: ${_ANS:=$_DEFAULT}
if [ "$_ANS" = "" ]; then
  case $_ANS in
  Y) return 0 ;;
  N) return 1 ;;
  esac
fi
  # user has selected something
    case $_ANS in
    y|Y|Yes|YES)
      return 0
      ;;
    n|N|No|NO)
      return 1
      ;;
    *) echo "Answer either Y or N, default is $_DEFAULT"
    ;;
    esac
  echo $_ANS
done
}

continue_prompt()
{
# continue_prompt
echo -n "Hit any key to continue.."
DUMMY=`read_a_char`
}

length_check()
{
# length_check
# $1=str to check length $2=max length
_STR=$1
_MAX=$2
_LENGTH=`echo $_STR |awk '{print length($0)}'`
if [ "$_LENGTH" -gt "$_MAX" ]; then
  return 1
else
  return 0
fi
}

a_number()
{
# a_number
# call: a_number $1=number
_NUM=$1
_NUM=`echo $1|awk '{if($0~/[^0-9]/) print "1"}'`
if [ "$_NUM" != "" ]
then
  # errors
  return 1
else
```

China-pub.com
下载


```
   return 0
 fi
 }
characters()
# characters
# to call: char_name string
{
_LETTERS_ONLY=$1
_LETTERS_ONLY=`echo $1|awk '{if($0~/[^a-zA-Z]/) print "1"}'`
if [ "$_LETTERS_ONLY" != "" ]
then
  # oops errors
  return 1
else
  # contains only chars
  return 0
fi
}

check_duplicate()
{
# check_duplicate
# check for employee number duplicate
# to call: check_duplicate string
_CODE=$1
MATCH=`grep "$_CODE\>" $DBFILE`
echo $_CODE
if [ "$MATCH" = "" ]; then
  return 0   # no duplicate
else
  return 1   # duplicate found
fi
}

add_rec()
{
# add_rec
# ==  STAFF NUMBER
while :
do
  echo -n "Employee Staff Number :"
  read NUM
  if [ "$NUM" != "" ]; then
    if a_number $NUM; then
      NUM_PASS=0
    else
      NUM_PASS=1
    fi
    if length_check $NUM 10; then
      LEN_PASS=0
    else
      LEN_PASS=1
    fi
# now check for duplicates...
if check_duplicate $NUM; then
```

China-pub.com 下载

```
    DUPLICATE=0
  else
    DUPLICATE=1
    echo "Staff Number: There is already a employee with this number"
    continue_prompt
  fi
  if [ "$LEN_PASS" = "0"  -a  "$NUM_PASS" = "0" -a "$DUPLICATE" = "0" ]
  then
    break
  else
    echo "Staff Number: Non-Numeric or Too Many Numbers In Field"
    continue_prompt
  fi
  else
    echo "Staff Number: No Input Detected, This Field Requires a Number"
    continue_prompt
  fi
done

# == First Name
while :
do
  echo -n  "Employee's First Name :"
  read F_NAME
  if [ "$F_NAME" != "" ]; then
    if characters $F_NAME; then
      F_NAME_PASS=0
    else
      F_NAME_PASS=1
    fi
    if length_check $F_NAME 20; then
      LEN_PASS=0
    else
      LEN_PASS=1
    fi
    # both conditons must be true to get out of this loop
    if [ "$LEN_PASS" = "0"  -a  "$F_NAME_PASS" = "0" ]; then
      break
    else
      echo "Staff First Name: Non-Character or Too Many Characters In
           Field"
      continue_prompt
    fi
  else
    echo "Staff First Name: No Input Detected, This Field Requires
         Characters"
    continue_prompt
  fi
done

# == Surname
while :
do
  echo -n "Employee's Surname    :"
  read S_NAME
```

```
    if [ "$S_NAME" != "" ]; then
      if characters $S_NAME; then
        S_NAME_PASS=0
      else
        S_NAME_PASS=1
      fi
      if length_check $S_NAME 20; then
        LEN_PASS=0
      else
        LEN_PASS=1
      fi
      if [ "$LEN_PASS" = "0"  -a  "$S_NAME_PASS" = "0" ]; then
        break
      else
        echo "Staff Surname: Non-Character or Too Many Characters In Field"
        continue_prompt
      fi
    else
      echo "Staff Surname: No Input Detected, This Field Requires Characters"
      continue_prompt
    fi
  done

  # == Department
  while :
  do
    echo -n  "Company Department    :"
    read DEPART
    case $DEPART in
      ACCOUNTS|Accounts|accounts) break;;
      SALES|Sales|sales)break;;
      IT|It|it) break;;
      CLAIMS|Claims|claims)break;;
      Services|SERVICES|services)break;;
      *) echo "Department: Accounts,Sales,IT,Claims,Services";;
    esac
  done
  }

  #  main
  clear
  echo -e "\t\t\tADD A EMPLOYEE RECORD"
  if [ -s $DBFILE ]; then :
  else
    echo "Information: Creating new file to add employee records"
    >$DBFILE
  fi
  add_rec
  if continue_promptYN "Do You wish To Save This Record " "Y"; then
    echo "$NUM:$F_NAME:$S_NAME:$DEPART" >>$DBFILE
    echo "record saved"

    sleep 1
    sort +2 -t: $DBFILE >$HOLD1 2> /dev/null
    if [ $? -ne 0 ]; then
```

China-pub.com 下载

```
      echo "problems trying to sort the file..check it out"
      exit 1
    fi
    mv $HOLD1 $DBFILE
    if [ $? -ne 0 ]; then
      echo "problems moving the temp sort file..check it out"
      exit 1
    fi

  else
    echo " record not saved"
    sleep 1
  fi
```

22.2 删除记录

要从文件中删除记录，首先要将记录传给用户以确保该记录是正确删除的记录。得到确认后才执行下列任务：

1) 查询记录。

2) 显示记录。

3) 确认删除。

4) 修改文件。

首先使用姓域查询记录，一旦从用户处得到需查询的姓，则使用 grep或awk进行处理，但是因为此文件不会有超过100个记录，将直接从文件中执行读取操作，进行匹配测试。

如果文件包括超过几百个记录，建议使用 awk，因为使用awk比直接从文件中读取数据快得多，同样也比用grep将各域分开存入变量要快一些。

可以使用awk或grep查询文件DBFILE：

```
echo "enter the surname to search "
read STR

# for awk use this
awk -F: '/$STR/' DBFILE

# for grep use this                    or
grep "$STR" DBFILE                     grep "$STR\>"DBFILE
```

注意使用awk时，变量用单引号括起来，如果不这样做，就不会返回任何数据。

将各域分开，并设置与其对应的变量（记住这里用冒号作分隔符）。必须改变IFS设置为冒号。如果不这样，就不能读取记录。改变 IFS设置时，最好先保存其设置，以便于脚本完成时再恢复它。

要保存IFS，使用：

```
SAVEDIFS=$IFS
```

将其改为冒号：

```
IFS=:
```

当用IFS完成操作后，简单的恢复它：

```
IFS=$SAVEDIFS
```

查询记录函数为get_rec，此函数不带参数。

```
get_rec()
{
# get_rec
clear
echo -n "Enter the employee surname :"
read STR
if [ "$STR" = "q" ]; then
  return 1
fi

REC=0
MATCH=no
if [ "$STR" != "" ]; then
  while read CODE F_NAME S_NAME DEPART
 do
   REC=`expr $REC + 1`
   tput cup 3 4
   echo -n " searching record.. $REC"
   if [ "$S_NAME" = "$STR" ]; then
     MATCH=yes
     display_rec
     break
   else
     continue
   fi
  done <DBFILE
else
  echo "Enter a surname to search for or q to quit"
fi
if [ "$MATCH" = "no" ]; then
  no_recs
fi
}
```

用户可以输入姓或q退出任务。一旦输入姓，执行测试以确保输入存在。比较好的测试方法是：

```
if ["$STR"!="";]then
```

然后是：

```
[-2 $STR]
```

第一个测试捕获只键入回车符的用户。第二个测试0长度字符串。

使用有意义的变量名从文件中读取各域，读取记录时使用计数，通过计数变化告之用户查询记录时发生某种动作。如果发现匹配模式，调用另一过程显示此域，用户然后使用 break命令跳出循环。如果未找到匹配模式，脚本进入下一循环步。找到匹配记录后，询问用户是否删除记录，缺省回答是no。

```
if continue_promptYN "Do You Wish To DELETE This Record" "N"; then
echo "DEL"
grep -v $STR DBFILE >$HOLD1 2> /dev/null
if [ $? -ne 0 ]; then
  echo "Problems creating temp file $HOLD1..check it out"
```

China-pub.com 下载

```
  exit 1
fi
...
```

使用grep -v执行记录删除，并使用字符串STR（STR保存用户删除的雇员姓）显示所有未匹配记录。

grep命令输出重定向到一临时工作文件中。然后文件移入初始 DBFILE中，删除后执行文件分类，输出重定向到一临时工作文件，然后再移回初始文件 DBFILE。临时工作文件可能首先被分类，然后再移回初始文件，而不是先移后分类。

使用最后状态命令执行测试所有文件移动操作。以下是删除一个记录的输出结果：

```
Enter the employee surname :Wilson

     searching record.. 6

 EMPLOYEE NO: 89232
 FIRST NAME  : Louise
 SURNAME     : Wilson
 DEPARTMENT  : Accounts

Do You Wish To DELETE This Record [Y..N] [N]:
```

删除记录的完整脚本如下：

```
$ pg dbase_del
#!/bin/sh
# dbase_del
# delete a record

# trap signals
trap "" 2 3 15

# DATAFILE
DBFILE=DBFILE

# temp files
HOLD1=HOLD1.$$
HOLD2=HOLD2.$$

continue_promptYN()
{
# continue_prompt
_STR=$1
_DEFAULT=$2
# check we have the right params
if [ $# -lt 1 ]; then
  echo "continue_prompt: I need a string to display"
  return 1
fi
while :
do
  echo -n "$_STR [Y..N] [$_DEFAULT]:"
read _ANS
: ${_ANS:=$_DEFAULT}
if [ "$_ANS" = "" ]; then
  case $_ANS in
  Y) return 0 ;;
  N) return 1 ;;
```

China-pub.com
下载


```
    esac
  fi
  case $_ANS in
  y|Y|Yes|YES)
    return 0
    ;;
  n|N|No|NO)
    return 1
    ;;
  *) echo "Answer either Y or N, default is $_DEFAULT"
    ;;
  esac
done
}

display_rec()
{
# display_rec
# could use cat << here document
tput cup 5 3
echo  "EMPLOYEE NO: $CODE"
echo  "FIRST NAME : $F_NAME"
echo  "SURNAME    : $S_NAME"
echo  "DEPARTMENT : $DEPART"
echo -e "\n\n"
}

no_recs()
{
# no_recs
echo  -e "\n\nSorry could not find a record with the name $STR"
}

get_rec()
{
# get_rec
clear
echo -n "Enter the employee surname :"
read STR
if [ "$STR" = "q" ]; then
  return 1
fi

REC=0
MATCH=no
if [ "$STR" != "" ]; then
  while read CODE F_NAME S_NAME DEPART
  do
    REC=`expr $REC + 1`
    tput cup 3 4
    echo -n " searching record.. $REC"
    if [ "$S_NAME" = "$STR" ]; then
      MATCH=yes
      display_rec
      break
    else
```

China-pub.com 下载

```
        continue
      fi
    done $DBFILE
  else
    echo "Enter a surname to search for or q to quit"
  fi
  if [ "$MATCH" = "no" ]; then
    no_recs
  fi
  }


  SAVEDIFS=$IFS
  IFS=:
  get_rec
  if [ "$MATCH" = "yes" ]; then
    if continue_promptYN "Do You Wish To DELETE This Record" "N"; then
      echo "DEL"
      grep -v $STR DBFILE >$HOLD1 2> /dev/null
      if [ $? -ne 0 ]; then
        echo "Problems creating temp file $HOLD1..check it out"
        exit 1
      fi
      mv $HOLD1 DBFILE
      if [ $? -ne 0 ]; then
        echo "Problems moving temp file..check it out"
        exit 1
      fi
      # sort the file after changes
      sort +2 -t: $DBFILE >$HOLD2 2> /dev/null
      if [ $? -ne 0 ]; then
        echo "problems trying to sort the file..check it out"
        exit 1
      fi
      mv $HOLD2 $DBFILE
      if [ $? -ne 0 ]; then
        echo "problems moving the temp sort file..check it out"
        exit 1
      fi
    else
      echo "no deletion"
      # no deletion
    fi # if wish to delete
  fi # if match

  # restore IFS settings
  IFS=$SAVEDIFS
```

22.3 修改记录

实际上已经编写了修改记录的大部分脚本，这些脚本在记录删除操作中。

找到正确记录后，设置所有该记录域变量到一临时工作文件，然后使用缺省设置变量：

```
:{default_variable=variable}
```

对于不想修改的域，简单输入回车键即可，然后缺省值放入临时工作变量中。在相关域内键入新值即可修改此域。

```
echo  -n -e "EMPLOYEE NO: $CODE\n"

echo  -n "FIRST NAME : [$F_NAME] >"
read _F_NAME
: ${_FNAME:=$F_NAME}
...
```

使用grep -v执行文件的实际修改操作。除了正在被修改的记录，所有记录重定向到一临时工作文件，这里雇员编号用作grep命令字符串参数：

```
grep -v $CODE $DBFILE >$HOLD1
```

然后提示用户是否保存记录。如果保存，重新修改的记录也写入临时工作文件，然后此临时工作文件移入原文件DBFILE中。

```
echo "$CODE:$_F_NAME:$_S_NAME:$_DEPART" >> $HOLD1
mv $HOLD1 $DBFILE
```

输出被重定向到一临时工作文件，然后将此文件分类，再移回原文件 DBFILE。最后状态命令测试文件移动操作。如果存在问题，告之用户。实际脚本中，在增加记录时执行的有效性测试也同样用于修改记录中。修改一个记录的输出结果如下：

```
Enter the employee surname :Penny

     searching record.. 7

EMPLOYEE NO: 98211
FIRST NAME : Simon
SURNAME    : Penny
DEPARTMENT : Services

Is this the record you wish to amend [Y..N] [Y]:
amending
EMPLOYEE NO: 98211
FIRST NAME : [Simon] >
SURNAME    : [Penny] >
DEPARTMENT : [Services] >Accounts
Ready to save this record [Y..N] [Y]:
```

完整脚本如下：

```
$ pg dbasechange
# !/bin/sh
# dbasechange
# amend a record

# ignore signals
trap "" 2 3 15

# temp files
DBFILE=DBFILE
HOLD1=HOLD1.$$
HOLD2=HOLD2.$$

continue_promptYN()
{
```

China-pub.com 下载

```
# continue_prompt
_STR=$1
_DEFAULT=$2
# check we have the right params
if [ $# -lt 1 ]; then
  echo "continue_prompt: I need a string to display"
  return 1
fi
while :
do
  echo -n "$_STR [Y..N] [$_DEFAULT]:"
  read _ANS
  : ${_ANS:=$_DEFAULT}
  if [ "$_ANS" = "" ]; then
    case $_ANS in
    Y) return 0 ;;
    N) return 1 ;;
    esac
  fi
  case $_ANS in
  y|Y|Yes|YES)
    return 0
    ;;
  n|N|No|NO)
    return 1
    ;;
  *) echo "Answer either Y or N, default is $_DEFAULT"
    ;;
  esac
done
}

display_rec()
{
# display_rec
# could use cat << as a here document, but I won't
tput cup 5 3
echo  "EMPLOYEE NO: $CODE"
echo  "FIRST NAME : $F_NAME"
echo  "SURNAME    : $S_NAME"
echo  "DEPARTMENT : $DEPART"
echo -e "\n\n"
}

no_recs()
{
# no_recs
echo  -e "\n\nSorry could not find a record with the name $STR"
}

get_rec()
{
# get_rec
clear
```

```
echo -n "Enter the employee surname :"
read STR
if [ "$STR" = "q" ]; then
  return 1
fi

REC=0
MATCH=no
if [ "$STR" != "" ]; then
  while read CODE F_NAME S_NAME DEPART
  do
    REC=`expr $REC + 1`
    tput cup 3 4
    echo -n " searching record.. $REC"
    if [ "$S_NAME" = "$STR" ]; then
      MATCH=yes
      display_rec
      break
    else
      continue
    fi
  done $DBFILE
else
  echo "Enter a surname to search for or q to quit"
fi
if [ "$MATCH" = "no" ]; then
  no_recs
fi
}

# main
SAVEDIFS=$IFS
IFS=:
get_rec
if [ "$MATCH" = "yes" ]; then
  if continue_promptYN "Is this the record you wish to amend" "Y"
  then
    echo "amending"
    # cannot change employee code
    echo  -n -e "EMPLOYEE NO: $CODE\n"

    echo  -n "FIRST NAME : [$F_NAME] >"
    read _F_NAME
    : ${_FNAME:=$F_NAME}

    echo  -n "SURNAME    : [$S_NAME] >"
    read _S_NAME
    : ${_S_NAME:=$S_NAME}

    echo  -n "DEPARTMENT : [$DEPART] >"
    read _DEPART
    : ${_DEPART:=$DEPART}

    grep -v $CODE $DBFILE >$HOLD1
```

China-pub.com 下载

```
if [ $? -ne 0 ]; then
  echo "Problems creating temporary file..check it out"
  exit 1
fi
     if continue_promptYN "Ready to save this record" "Y"; then
       echo "$CODE:$_F_NAME:$_S_NAME:$_DEPART" >> $HOLD1
       mv $HOLD1 $DBFILE
       if [ $? -ne 0 ]; then
         echo "Problems moving temporary file...check it out"
       fi
       echo " Record Amended"
       # now sort the file after changes,
       # could have been done in one file movement of course
       sort +2 -t: $DBFILE >$HOLD2 2> /dev/null
       if [ $? -ne 0 ]; then
         echo "problems trying to sort the file..check it out"
         exit 1
       fi
       mv $HOLD2 $DBFILE
       if [ $? -ne 0 ]; then
         echo "problems moving the temp sort file..check it out"
         exit 1
       fi


     else #if amend aborted
       echo "Amend aborted"
       exit 0
     fi
   else # if no amend
     echo "no amending"
     # no deletion
   fi # if wish to delete
fi # if match


IFS=$SAVEDIFS
```

22.4 查看记录

用户可能要查看所有记录或其中一部分。如果查看所有记录，使用 cat命令和awk，如果记录包含很多域，那么很有必要定量显示输出结果，使其对用户更加实用。

```
if [ "$STR" = "all" ]; then
  echo "Surname   Name   Employee Code"
  echo "___________________________________"
  cat $DBFILE |awk -F: '{print $2"\t"$3"\t\t"$1}' | more
  return 0
fi
```

在删除和修改记录中，已经讲过了怎样显示单一记录，用户有选择的查看记录选项时唯一增加的功能就是打印一个记录。以下脚本段将记录发往打印机：

```
pr  <<- MAYDAY

RECORD No            : $REC
EMPLOYEE NUMBER      : $CODE
```

```
EMPLOYEE NAME       : $F_NAME
EMPLOYEE SURNAME    : $S_NAME
EMPLOYEE DEPARTMENT : $DEPART
MAYDAY
```

查看记录时输出结果如下：

```
Enter the employee surname to view or all for all records:Wilson

     searching record.. 8

EMPLOYEE NO: 89232
FIRST NAME : Peter
SURNAME    : Wilson
DEPARTMENT : IT

Do You Wish To Print This Record [Y..N] [N]:
```

查看记录的完整脚本如下：

```
$ pg dbaseview
#!/bin/sh
# dbaseview
# view records

# ignore signals
trap "" 2 3 15

# temp files
HOLD1=HOLD1.$$
DBFILE=DBFILE

continue_promptYN()
{
# continue_prompt
_STR=$1
_DEFAULT=$2
# check we have the right params
if [ $# -lt 1 ]; then
  echo "continue_prompt: I need a string to display"
  return 1
fi
while :
do
  echo -n "$_STR [Y..N] [$_DEFAULT]:"
  read _ANS
  : ${_ANS:=$_DEFAULT}
  if [ "$_ANS" = "" ]; then
    case $_ANS in
    Y) return 0 ;;
    N) return 1 ;;
    esac
  fi
  case $_ANS in
  y|Y|Yes|YES)
    return 0
    ;;
  n|N|No|NO)
    return 1
```

China-pub.com 下载

```
    ;;
  *) echo "Answer either Y or N, default is $_DEFAULT"
    ;;
  esac
done
}

display_rec()
{
# display_rec
# could use cat <<.
tput cup 5 3
echo  "EMPLOYEE NO: $CODE"
echo  "FIRST NAME : $F_NAME"
echo  "SURNAME    : $S_NAME"
echo  "DEPARTMENT : $DEPART"
echo -e "\n\n"
}

no_recs()
{
# no_rec
echo  -e "\n\nSorry could not find a record with the name $STR"
}

get_rec()
{
# get_rec
clear
echo -n "Enter the employee surname to view or all for all records:"
read STR
if [ "$STR" = "q" ]; then
  return 1
fi
if [ "$STR" = "all" ]; then
  # view all recs
  echo "Surname   Name   Employee Code"
  echo "_______________________________"
  cat $DBFILE |awk -F: '{print $2"\t"$3"\t\t"$1}' | more
  return 0
fi
REC=0
MATCH=no
if [ "$STR" != "" ]; then
  while read CODE F_NAME S_NAME DEPART
  do
    REC=`expr $REC + 1`
    tput cup 3 4
    echo -n " searching record.. $REC"
    if [ "$S_NAME" = "$STR" ]; then
      # found name
      MATCH=yes
      display_rec
      break
    else
      continue
```

China-pub.com
下载


```
    fi
  done <$DBFILE
else
  echo "Enter a surname to search for or q to quit"
fi
if [ "$MATCH" = "no" ]; then
  no_recs
fi
}

# main
SAVEDIFS=$IFS
IFS=:
get_rec
if [ "$MATCH" = "yes" ]; then
  if continue_promptYN "Do You Wish To Print This Record" "N"; then
  lpr  <<- MAYDAY

  RECORD No: $REC

  EMPLOYEE NUMBER      : $CODE
  EMPLOYEE NAME        : $F_NAME
  EMPLOYEE SURNAME     : $S_NAME
  EMPLOYEE DEPARTMENT  : $DEPART

  MAYDAY
  else
    echo "No print of $S_NAME"
    # no print
  fi # if wish to print
fi # if match

IFS=$SAVEDIFS
```

22.5 小结

验证用户输入的有效性很重要，也是一种高级技巧。虽然你可能有时知道记录接受输入的内容，但用户通常并不知道。

在计算机工业发展史上有一句老话：送进的是垃圾，出来的肯定是垃圾，但知道这一点时已经太晚了，意即如果没有在脚本中测试垃圾数据，就会输出垃圾信息。

China-pub.com
下载

第23章 调试脚本

shell编程最烦人的一项工作是调试问题。有一些方法可以借鉴，但是最好能在问题出现前防止大部分错误，为此应遵循以下规则。

将设计脚本分成几个任务或过程，然后在继续下一步前分别予以测试。

本章内容有：

- 一般错误。
- set命令介绍。

没有比在脚本中发现一个难以察觉的错误更令人头疼的了，然而，随着编程经验不断丰富，查询手段也相应增加。

经常碰到的问题是忘了使用引号或在 if语句末尾未加fi。

需要牢记的一点是当 shell打印出一个脚本错误后，不要只看那些疑问行。而是要观察整个相关代码段。shell不会对错误进行精确定位，而是在试图结束一个语句时进行错误统计。

23.1 一般错误

23.1.1 循环错误

for、while、until和case语句中的错误是指实际语句段不正确。也许漏写了固定结构中的一个保留字。

下面错误打印信息 done，这是一个很好的线索。因为这时知道正在处理一个 while语句。回溯脚本段，检查while语句，是否漏写或错写了关键字，如 do或者正在使用的条件语句。

```
syntax error near unexpected token ‘done’
line 31: ‘ done’
```

23.1.2 典型的漏写引号

第二个典型错误是漏写引号错误。经常要注意这个问题，因为此错误经常出现。这里给出解决这类错误的唯一方案是在脚本中确保所有引号成对出现。

```
unexpected EOF while looking for ‘"’
line 36: syntax error
```

当shell打印出错误行后，通常在 vi编辑器中查看文件。使用 vi的set nu选项调试错误，先进入vi，然后点击<ESC>键，后跟一冒号，再键入 set nu <return>，这时给出文本行号，然后进入shell打印错误行。

23.1.3 测试错误

另一个常见错误是在使用 -eq语句时忘记在测试条件一边使用数字取值。

如果得到下列错误提示，通常是由于两件事情：需要在变量和方括号间加空格；在方括号里漏写操作符。

```
[: missing ']'
```

23.1.4 字符大小写

经验上讲大多数错误是由于使用变量时大小写保持不一致。例如经常在开始定义时用大写，然后在变量调用时用了小写字符，这样难免变量会没有赋值。

23.1.5 for循环

使用for循环时，有时会忘了在循环的列表部分用 $符号，特别是在读取字符串时。

23.1.6 echo

最有用的调试脚本工具是echo命令。一般在可能出现问题的脚本重要部分加入 echo命令，例如在变量读取或修改操作其前后加入 echo命令。

使用最后状态命令判断命令是否成功，这里需要注意的是，不要使用 echo命令后直接加最后状态命令，因为此命令永远为真。

23.2 set命令

set命令可辅助脚本调试。以下是set命令常用的调试选项：

set -n 读命令但并不执行。

set -v 显示读取的所有行。

set -x 显示所有命令及其参数。

将set选项关闭，只需用+替代-。有人总认为+应该为开，而-应为关闭，但实际刚好相反。

可以在脚本开始时将 set选项打开，然后在结束时关闭它。或在认为有问题的特殊语句段前后打开及关闭它。

下面看一个例子。以下脚本将名字保存在变量列表中。用户输入名字， for循环循环变量列表查看是否有匹配模式。注意这里在脚本开始时使用了 set -x，并在结尾部分关闭它。

```
$ pg error
#!/bin/sh
# error
# set set -x
set -x
LIST="Peter Susan John Barry Lucy Norman Bill Leslie"
echo -n "Enter your Name :"
read NAME

for LOOP in $LIST
do
  if [ "$LOOP" = "$NAME" ]; then
    echo "you're on the list, you're in"
    break
  fi
done
# unset set -x
set +x
```

运行此脚本，给出一个不在列表中的名字，输出如下：

China-pub.com 下载

```
$ error
error
+ error
+ LIST=Peter Susan John Barry Lucy Norman Bill Leslie
+ echo -n Enter your Name :
Enter your Name :+ read NAME
Harry
+ [ Peter = Harry ]
+ [ Susan = Harry ]
+ [ John = Harry ]
+ [ Barry = Harry ]
+ [ Lucy = Harry ]
+ [ Norman = Harry ]
+ [ Bill = Harry ]
+ [ Leslie = Harry ]
```

输出显示对变量列表进行循环时所有的比较操作。当读取文件或进行字符串或取值的比较发现问题时，使用set命令是很有必要的。

23.3 小结

跟踪错误的最好方式是亲自查阅脚本，并使用 set命令并加大量的echo语句。

China-pub.com 下载

第24章 shell嵌入命令

实际上已经用过了许多 shell嵌入命令。可能要想什么是 shell嵌入，这些命令是在实际的Bourne shell里创建而不是存在于/bin或usr/bin目录里。嵌入命令比系统里的相同命令要快。

本章内容有：

• 标准的Bourne shell嵌入命令列表

例如，cd和pwd命令可同时在系统和嵌入命令中发现。如果要运行系统版，简单输入命令路径即可：

```
/bin/pwd
```

24.1 shell嵌入命令完整列表

表24-1给出标准嵌入命令的完整列表。

表24-1 标准嵌入命令

:	空，永远返回为true
.	从当前shell中执行操作
break	退出for、while、until或case语句
cd	改变到当前目录
continue	执行循环的下一步
echo	反馈信息到标准输出
eval	读取参数，执行结果命令
exec	执行命令，但不在当前shell
exit	退出当前shell
export	导出变量，使当前shell可利用它
pwd	显示当前目录
read	从标准输入读取一行文本
readonly	使变量只读
return	退出函数并带有返回值
set	控制各种参数到标准输出的显示
shift	命令行参数向左偏移一个
test	评估条件表达式
times	显示shell运行过程的用户和系统时间
trap	当捕获信号时运行指定命令
ulimit	显示或设置shell资源
umask	显示或设置缺省文件创建模式
unset	从shell内存中删除变量或函数
wait	等待直到子进程运行完毕，报告终止

下面讲述一些未涉及或前面讲解不深的命令。

24.1.1 pwd

显示当前目录

China-pub.com 下载

```
$ pwd
/tmp
```

24.1.2 set

在查看调试脚本、打开或关闭shell选项时，曾用到set命令。set也可用于在脚本内部给出其运行参数，以下举例说明。假定有一段脚本控制两个参数，但并不向脚本传递参数而是在脚本内部设置其取值。可以用set命令完成此功能。

格式为：

```
set param1 param2 ..
```

下面的脚本设置参数为accounts.doc和accounts.bak，然后对参数进行循环处理。

```
$ pg set_ex
#!/bin/sh
set accounts.doc accounts.bak

while [ $# != 0 ]
do
  echo $1
  shift
done

$ set_ex
accounts.doc
accounts.bak
```

当测试一段脚本且脚本包含参数时，这样使用 set命令有很多用处。其一就是不必在每次运行脚本时重复输入参数。

24.1.3 times

times命令给出用户脚本或任何系统命令的运行时间。第一行给出 shell消耗时间，第二行给出运行命令消耗的时间。下面是times命令的输出结果：

```
$ times
0m0.10s 0m0.13s
0m0.49s 0m0.36s
```

相信以后会经常用到该命令。

24.1.4 type

使用type查询命令是否仍驻留系统及命令类型。 type打印命令名是否有效及该命令在系统的位置。例如：

```
$ type mayday
type: mayday: not found
$ type pwd
pwd is a shell builtin
$ type times
times is a shell builtin
$ type cp
cp is /bin/cp
```

24.1.5 ulimit

ulimit设置运行在shell上的显示限制。通常此命令定位于文件 /etc/profile中，但是可以从当前shell或用户.profile文件中将之移入用户需要的位置。ulimit一般格式如下：

```
ulimit options
```

ulimit有几个选项，以下是一些常用的选项：

选 项	含 义
-a	显示当前限制
-c	限制内核垃圾大小
-f	限制运行进程创建的输出文件的大小

例如ulimit取值为：

```
$ ulimit -a
core file size (blocks)   1000000
data seg size (kbytes)    unlimited
file size (blocks)        unlimited
max memory size (kbytes)  unlimited
stack size (kbytes)       8192
cpu time (seconds)        unlimited
max user processes        256
pipe size (512 bytes)     8
open files                256
virtual memory (kbytes)   2105343
```

将内核文件垃圾数目设置为0：

```
$ ulimit -c 0
$
$ ulimit -a
core file size (blocks)   0
data seg size (kbytes)    unlimited
file size (blocks)        unlimited
max memory size (kbytes)  unlimited
stack size (kbytes)       8192
cpu time (seconds)        unlimited
max user processes        256
pipe size (512 bytes)     8
open files                256
virtual memory (kbytes)   2105343
```

24.1.6 wait

wait命令等待直到一个用户子进程完成，可以在wait命令中指定进程ID号。如果并未指定，则等待直到所有子进程完成。

等待所有子进程运行完毕：

```
$ wait
```

24.2 小结

上面讲述了shell嵌入命令，其中大部分前面已经用过，本章仅详细讲述了其使用方法。

China-pub.com
下载

第五部分　高级shell编程技巧

第25章　深入讨论<<

我们在介绍标准输入和标准输出以及 while循环的时候已经几次遇到 <<的应用。我们学会了如何发送邮件，如何构建一个菜单，不过 <<还有很多其他的用法。

本章将介绍以下内容：

• 快速创建一个文件。

• 自动进入菜单。

• ftp传输。

• 连接至其他应用系统。

该命令的一般形式为：

```
command <<word
text
word
```

这里再简要回顾一下 <<的用法。当shell看到<<的时候，它就会知道下一个词是一个分界符。在该分界符以后的内容都被当作输入，直到 shell又看到该分界符(位于单独的一行)。这个分界符可以是你所定义的任何字符串。

可以使用<<来创建文件、显示文件列表、排序文件列表以及创建屏幕输入。

25.1　快速创建一个文件

可以使用这种方法快速创建一个文件，并向其中存入一些文本：

```
$ cat >> myfile <<NEWFILE
```

现在可以输入一些文本，结束时只要在新的一行键入 NEWFILE即可，这样就创建了一个名为myfile的文件，该文件中包含了一些文本。

如果打开了一个已经存在的文件，输入的内容会附加到该文件的末尾。

如果使用tab键，注意，一些老版本的 shell可能无法正确理解它的含义。为了解决这一问题，可以在<<之后加一个横杠-，就像下面这样：

```
cat >> myfile <<- NEWFILE
...
```

25.2　快速创建打印文档

假如希望打印一小段信息，可以采用这种方法而不必使用 vi编辑器。在本例中，一旦在输入QUICKDOC之后按回车键，相应的文档就会被送到打印机。

```
$ lpr <<QUICKDOC
   **** INVITATION****
```

China-pub.com 下载

```
 The Star Trek convention is in town
next week. Be there.

Ticket prices: (please phone)
--------------------------------
QUICKDOC
```

25.3 自动选择菜单

不但可以很方便地使用 <<创建菜单屏幕，还可以使用它来自动选择菜单，而不是由用户手工进行选择。

我编写了一个菜单驱动的数据库管理脚本，可以使用它来完成备份和其他系统管理任务。该脚本本来是在白天由用户来运行的，现在决定把这些工作交给 cron夜间完成，我不想再另外写一个自动运行的脚本，于是我使用 <<中的输入来选择syb_backup脚本的菜单选项。下面介绍一下该脚本的菜单。

主菜单如下，选择2：

```
1: Admin Tasks
2: Sybase Backups
3: Maintenance Tasks
Selection > 2
```

第二层菜单如下，选择3：

```
1: Backup A Single Database
2: Backup Selected Databases
3: Backup All Databases
Selection > 3
```

第三级菜单如下，选择Y：

```
1. dw_levels
2. dw_based
3. dw_aggs
...
...
```

从菜单来看，如果要备份所有的数据库，需要键入：

1) 菜单脚本的名字， syb_backup。

2) 键入2。

3) 键入3。

4) 键入Y。

下面的脚本能够自动运行数据库备份脚本 syb_backup：

```
$ pg auto.sybackup
#!/bin/sh
# set the path
PATH=/usr/bin:/usr/sbin:/sybase/bin:$LOCALBIN
export PATH
# set the sybase variable
DSQUERY=COMET; export DSQUERY
# set the TERM and init it
TERM=vt220; export TERM
```

China-pub.com
下载


```
tput -T vt220 init
#  keep a log of all output
log_f=/logs/sql.backup.log
#
>$log_f

# here's the code that does all the work !
/usr/local/sybin/syb_backup >> $log_f 2>&1  << MAYDAY
2
3
Y
MAYDAY

chown sybase $log_f
```

该脚本中的重定向部分是：

```
usr/local/sybin/syb_backup >> $log_f 2>&1  << MAYDAY
2
3
Y
MAYDAY
```

让我们来分析一下这一部分，这里给出了脚本syb_backup的全路径；>>$log_f 2>&1意味着所有的输出都重定向到$log_f中，该变量的值为/logs/sql.backup.log。这是一个良好的习惯，因为这样就能够捕捉到所运行的程序或脚本的所有输出，如果出现错误的话，也能够被记录下来。

<<MAYDAY之后的内容就是手工运行syb_backup脚本所需要输入的内容，直到遇到另外一个MAYDAY结束。

这样，我就不需要重新再写一个脚本；如果已经有一个菜单驱动的脚本，只需再编写一个使用<<输入的脚本就可以自动运行原先的脚本。

25.4 自动ftp传输

<<的另外一个流行的应用就是自动ftp传输。在使用ftp时，如果能够向用户提供一个简单的界面就好了。下面的脚本使用了匿名用户 anonymous建立了一个ftp连接。这是一个特殊的用户，它使得系统能够创建一个含有公共目录的安全帐户。一般来说，所有以匿名用户身份进行连接的用户都只能从公共目录中下载文件，不过只要权限允许，用户也可以上载。

匿名用户的口令可以是任何字符串，不过最好使用主机名加上本地用户名，或电子邮件地址。

下面的脚本将会提示如下的信息：

1) 希望登录的远程主机。

2) 文件传输的类型是二进制方式还是ASCII方式。

3) 要下载的文件名。

4) 存放下载文件的本地目录。

当用户输入想要连接的主机之后，首先执行一个名为 traceroute的脚本验证本地主机是否能够连接到远程主机。如果traceroute执行失败，这个自动ftp传输的脚本将会再次提示用户输入主机名。

China-pub.com 下载

用户在看到传输模式选择的提示之后按回车键，将会选择缺省的二进制模式。

用户在输入所要下载的文件名之后，将会被提示输入保存下载文件的本地目录。缺省的本地目录是/tmp。如果用户所给出的目录无法找到，仍将使用缺省的 /tmp目录。

下载文件在本地的文件名将是原文件名加上 .ftp后缀。

最后，用户所有的选择都将在屏幕上显示出来，待用户确认后开始进行传输。

下面就是该脚本运行时在屏幕上的显示：

```
$ ftpauto
User: dave              05/06/1999                 This host: bumper
                FTP RETRIEVAL / POSTING SCRIPT
                ==============================
                      Using the ID of anonymous
Enter the host you wish to access :uniware
Wait..seeing if uniware is out there..
bumper can see uniware
What type of transfer / receive mode ?
 1 : Binary
 2 : ASCII
Your choice [1..2] [1]:
   Enter the name of the file to retrieve :gnutar.Z
 Enter the directory where the file is to be placed[/tmp] :
                Host to connect is: uniware
                File to get is     : gnutar.Z
                Mode to use is     : binary
                File to be put in  : /tmp/gnutar.Z.ftp
          Ready to get file 'y' or 'q' to quit? [y..q] :
```

下面就是该脚本的内容：

```
$ pg ftpauto
#!/bin/sh
# ftp script
# ftpauto
USER=`whoami`
MYDATE=`date +%d/%m/%Y`
THIS_HOST=`hostname -s`
tracelog=/tmp/tracelog.$$

while :
do
  # loop forever
  tput clear
  cat <<MAYDAY
  User: $USER              $MYDATE                 This host: $THIS_HOST
                     FTP RETRIEVAL SCRIPT
                     ====================
                   Using the ID of anonymous
  MAYDAY
  echo -n "Enter the host you wish to access :"
  read DEST_HOST
  # is a hostname entered ???
  if [ "$DEST_HOST" = "" ]
  then
    echo "No destination host entered" >&2
    exit 1
  fi
```

China-pub.com 下载


```
  # can we see the host ???
  echo "Wait..seeing if $DEST_HOST is out there.."
  # use traceroute to test connectivity
  traceroute $DEST_HOST > $tracelog 2>&1

  if grep "unknown host" $tracelog  >/dev/null 2> then
    echo "Could not locate $DEST_HOST"
    echo -n "Try another host? [y..n] :"
    read ANS
    case $ANS in
    y|Y) ;;
    *) break;;   # get out of the forever loop
    esac
  else
    echo "$THIS_HOST can see $DEST_HOST"
    break   # get out of the forever loop
  fi
done

# the default is binary
echo  "What type of transfer /receive mode ?"
echo  " 1 : Binary"
echo  " 2 : ASCII"
echo -n -e "\fYour choice [1..2] [1]:"
read $TYPE
case $TYPE in
1) MODE=binary
  ;;
2) MODE=ascii
  ;;
 *) MODE=binary
  ;;
 esac

 echo -n "   Enter the name of the file to retrieve :"
 read FILENAME
 if [ "$FILENAME" = "" ]; then
   echo "No filename entered" >&2
   exit 1
 fi

 # default is tmp
 echo -n -e "\f Enter the directory where the file is to be placed[/tmp] :"
 read DIREC
 cd $DIREC >/dev/null 2>&1
 # if we cannot cd to the directory then use tmp
 if [ "$DIREC" = "" ]; then
   DIREC=/tmp
 fi

 if [ $? != 0 ]
 then
   echo "$DIREC does not exist placing the file in /tmp anyway"
   DIREC=/tmp
 fi
```

China-pub.com 下载

```
echo -e "\t\tHost to connect is: $DEST_HOST"
echo -e "\t\tFile to get is     : $FILENAME"
echo -e "\t\tMode to use is     : $MODE"
echo -e "\t\tFile to be put in : $DIREC/$FILENAME.ftp"
echo -e -n "\t\tReady to get file 'y' or 'q' to quit? [y..q] :"
read ANS
case $ANS in
Y|y);;
q|Q) exit 0;;
*) exit 0 ;;
esac
echo "ftp.."
ftp -i -n $DEST_HOST<<FTPIT
user anonymous $USER@$THISHOST
$MODE
get $FILENAME $DIREC/$FILENAME.ftp
quit
FTPIT
if [ -s $DIREC/$FILENAME.ftp ]
then
  echo "File is down"
else
  echo "Unable to locate $FILENAME.ftp"
fi
```

在ftp命令中使用<<时，使用了ftp -i -n选项，这意味着不要自动登录，而且关闭交互模式。这样就使得脚本可以使用 user命令进行登录。口令是 $USER@$THISHOST，在这里就是dave@bumper。

如果用户每天从同一台主机上下载相同的文件，比如说是包含前一天销售数据的文件，那么用户就没有必要每天都输入同样的主机名和文件名。可以设置 DEST_HOST和FILENAME变量的缺省值，这样就可以使用户不必每天都输入同样的主机名和文件名。

下面是ftp自动传输脚本中提示用户输入主机名的一段，但是现在不同的是， DEST_HOST变量已设置了缺省值my_favourite_host。现在用户可以另外输入一个不同的主机名，也可以敲回车键选择缺省值。

注意，现在不必再检查用户是否输入了一个值，因为如果用户没有输入的话，该变量将被赋予缺省值。

```
echo -n "Enter the host you wish to access :"
read DEST_HOST
: ${DEST_HOST:="my_favourite_host"}
echo "Wait..seeing if $DEST_HOST is out there.."
traceroute $DEST_HOST >$tracelog 2>&1
...
```

25.5 访问数据库

shell脚本一个常用的用途就是访问数据库系统获得信息。实现这样的功能， <<是再理想不过了。可以用它来输入你在面对数据库提示时所做的各种选择。下面的例子并不是数据库中的一个练习，而是为了用来介绍如何使用 <<来连接其他应用程序，完成相应的任务。

对于某一个数据库系统来说，在使用某种第三方产品进行访问时， select into功能将会被

关闭。这意味着该数据库不能被用来插入数据或创建临时表。

为了解决这个问题，我们使用 <<进行数据库连接，并使用一个 for循环来提供各个数据库名，一旦连接成功， <<将用来向sql命令提供选项。

下面就是该脚本。

```
$ pg set.select
#!/bin/sh
# set.select
# fixes known bug. Sets the select into db option on
PATH=$PATH:/sybase/bin:/sybase/install
export PATH
SYBASE="/sybase"; export SYBASE
DSQUERY=ACESRV; export DSQUERY
PASSWORD="prilog"
DATABASES="dwbased tempdb aggs levels reps accounts"

for loop in $DATABASES
do
  su sybase -c '/sybase/bin/isql -Usa -P$PASSWORD' << MAYDAY
  use master
  go
  sp_dboption $loop,"select into/bulkcopy/pllsort",true
  go
  use $loop
  go
  checkpoint
  go
  MAYDAY
done
```

让我们来看一看使用 <<的部分， shell在执行了变量替代以后将运行下面的一段命令。

```
use master
go
sp_dboption dwbased,"select into/bulkcopy/pllsort",true
  go
  use dw_based
  go
  checkpoint
  go
```

当shell看到结束的分界符 MAYDAY时，该脚本将开始下一次循环，对另外一个数据库进行操作。下面就是运行的结果：

```
$ set.select
Database option 'select into/bulkcopy/pllsort' turned ON for database
'dwbased'.
Run the CHECKPOINT command in the database that was changed.
(return status = 0)
Database option 'select into/bulkcopy/pllsort' turned ON for database
'tempdb'.
Run the CHECKPOINT command in the database that was changed.
(return status = 0)
Database option 'select into/bulkcopy/pllsort' turned ON for database
'aggs'.
Run the CHECKPOINT command in the database that was changed.
```

China-pub.com 下载

```
(return status = 0)
```

25.6 小结

本章进一步给出了一些使用 <<来自动完成某些任务的例子。 <<的用途很广，特别是在连接某些应用程序或使用 ftp时。你可以灵活地使用 <<来自动运行以前编写的脚本，从而完成各种不同的任务。

China-pub.com
下载

第26章　shell　工　具

本章将讨论以下内容：

• 创建以日期命名的文件及临时文件。
• 信号。
• trap命令以及如何捕获信号。
• eval命令。
• logger命令。

26.1　创建保存信息的文件

任何脚本都应该能够创建临时文件或日志文件。在运行脚本做备份时，最好是保存一个日志文件。这些日志文件通常在文件系统中保留几周，过时将被删除。

在开发脚本的时候，可能总要创建一些临时的文件。在正常运行脚本的时候，也要使用临时文件保存信息，以便作为另外一个进程的输入。可以使用 cat命令来显示一个临时文件的内容或把它打印出来。

26.1.1　使用date命令创建日志文件

在创建日志文件时，最好能够使它具有唯一性，可以按照日志文件创建的日期和时间来识别这些文件。我们可以使用 date命令做到这一点。这样就能够使日期和时间成为日志文件名中的一部分。

为了改变日期和时间的显示格式，可以使用如下的命令：

```
date option + %format
```

使用加号‘＋’可以设置当前日期和时间的显示格式。下面的例子将日期以日、月、年的格式显示：

```
$ date +%d%m%y
090699
```

下面是一些常用的日期格式：

```
$ date +%d-%m-%y
09-06-99
```

```
$ date +%A%e" "%B" "%Y
Wednesday 9 June 1999
```

下面的命令可以使时间按照hh:mm的格式显示：

```
$ date +%R
10:07
```

```
$ date +%A" "%R" "%p
Wednesday 10:09 AM
```

下面的命令可以显示完整的时间：

```
$ date +%T
```

China-pub.com 下载

```
10:29:41

$ date +%A" "%T
Wednesday  10:31:19
```

注意，如果希望在日期和时间的显示中包含空格，要使用双引号。

在文件名中含有日期的一个简单办法就是使用置换。把含有你所需要的日期格式的变量附加在相应的日志文件名后面即可。

在下面的例子中我们创建了两个日志文件，一个使用了 dd，mm，yy的日期格式，另一个使用了dd，hh，mm的时间格式。

下面就是这个脚本。

```
$ pg log
#!/bin/sh
# log
#
MYDATE=`date +%d%m%y`
# append MYDATE to the variable LOGFILE that holds the actual filename of
the log.
LOGFILE=/logs/backup_log.$MYDATE
# create the file
>$LOGFILE

MYTIME=`date +%d%R`
LOGFILE2=/logs/admin_log.$MYTIME
# create the file
>$LOGFILE2
```

运行上面的脚本后，得到这样两个日志文件。

```
backup_log.090699
admin_log.0910:18
```

26.1.2 创建唯一的临时文件

在本书的前面讨论特殊变量时，曾介绍变量 $$，该变量中保存有你所运行的当前进程的进程号。可以使用它在我们运行的脚本中创建一个唯一的临时文件，因为该脚本在运行时的进程号是唯一的。我们只要创建一个文件并在后面附加上 $$即可。在脚本结束时，只需删除带有$$扩展的临时文件即可。Shell将会把$$解析为当前的进程号，并删除相应的文件，而不会影响以其他进程号做后缀的文件。

在命令行中输入如下的命令：

```
$ echo $$
281
```

这就是当前的进程号，如果你执行这个命令，看到的结果可能会有所不同。现在如果我创建另一个登录进程并输入同样的命令，将会得到一个不同的进程号，因为我已经启动了一个新的进程。

```
$ echo $$
382
```

下面的例子中，创建了两个临时文件，并进行了相应的操作，最后在结束时删除了这些文件。

```
$ pg tempfiles
#!/bin/sh
# tempfiles
# name the temp files
HOLD1=/tmp/hold1.$$
HOLD2=/tmp/hold2.$$

# do some processing using the files
df -tk >$HOLD1
cat $HOLD1 >$HOLD2
# now delete them
rm /tmp/*.$$
```

当上面的脚本运行时，将会创建这样两个文件：

hold1.408
hold2.408

在执行rm /tmp/*.$$时，shell实际上将该命令解析为rm /tmp/*.408。

记住，该进程号只在当前进程中唯一。例如，如果我再次运行上面的脚本，将会得到一个新的进程号，因为我已经创建了一个新的进程。

如果文件有特殊用途的话，那么创建含有日期的文件，就可以使你很容易地查找到它们。而且还可以很容易地按照日期删除文件，因为这样一眼就能看出哪个文件是最新的，哪个文件是最“旧”的。

还可以使用这种方法来快速地创建临时文件，它们在当前进程中是唯一的。在脚本结束之前，也很容易删除这些临时文件。

26.2 信号

信号就是系统向脚本或命令发出的消息，告知它们某个事件的发生。这些事件通常是内存错误，访问权限问题或某个用户试图停止你的进程。信号实际上是一些数字。下表列出了最常用的信号及它们的含义。

信 号	信 号 名	含 义
1	SIGHUP	挂起或父进程被杀死
2	SIGINT	来自键盘的中断信号，通常是 <CTRL-C>
3	SIGQUIT	从键盘退出
9	SIGKILL	无条件终止
11	SIGSEGV	段（内存）冲突
15	SIGTERM	软件终止（缺省杀进程信号）

还有信号0，我们前面在创建.logout文件时已经遇到过。该信号为“退出shell”信号。为了发出信号0，只要从命令行键入exit，或在一个进程或命令行中使用<CTRL-D>即可。

发送信号可以使用如下的格式：

```
kill [-signal no:| signal name] process ID
```

使用kill命令时不带任何信号或名字意味着使用缺省的信号15。

可以使用如下的命令列出所有的信号：

```
$ kill -l
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
```

China-pub.com 下载

```
 5) SIGTRAP      6) SIGIOT       7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
30) SIGPWR
```

26.2.1 杀死一个进程

发送信号1将使一个进程重新读入配置文件。例如，你在运行域名服务（DNS）守护进程named，现在你对域名数据库文件做了某些修改，这时不需要杀死该守护进程再重新启动，只需使用kill -1命令向其发送信号1。Named进程将重新读入它的配置文件。

下面的例子向系统中一个名为mon_web的进程发送信号9（无条件终止）来杀死它。首先使用ps命令得到相应的进程号。

```
$ ps -ef | grep mon_web |grep -v root
157  ?  S    0:00 mon_web
```

如果系统不支持ps -ef命令，那么可以使用ps xa。为了杀死该进程，我可以使用下面的两种方法之一：

```
kill -9 157
```

或

```
kill -s SIGKILL 157
```

在有些系统中，不必使用-s，例如：kill SIGKILL 157。

下面的脚本将根据进程名来杀死一个进程，拟被杀死的进程名作为该脚本的一个参数。在执行相应的命令之前，将会首先检查是否存在这样的进程。在这里使用 grep命令来匹配相应的进程名。如果匹配成功，则向用户提示进程已经找到，并询问用户是否杀死该进程。最后使用kill -9命令杀死相应的进程。

下面就是该脚本。

```
$ pg pskill
#!/bin/sh
# pskill
HOLD1=/tmp/hold1.$$
PROCESS=$1
usage()
{

# usage
echo "Usage :`basename $0` process_name"
exit 1
}
if [ $# -ne 1 ]; then
  usage
fi

case $1 in
*)
# grep the process, do not include our script in the output from ps
# extract fields 1 and 6, redirect to a temp file
```

China-pub.com 下载


```
  ps x | grep $PROCESS | grep -v $0 | awk '{print $1"\t" $6}'>$HOLD1
  # ps -ef |..     if ps x does not work
  ;;
esac

# is the file there??
if [ ! -s $HOLD1 ]; then
  echo "No processes found..sorry"
  exit 1
fi

# read in the contents from the temp file and display the fields
while read LOOP1 LOOP2
do
  echo $LOOP1 $LOOP2
done <$HOLD1
echo -n "Are these the processes to be killed ? [y..n] >"
read ANS

case $ANS in
Y|y) while read LOOP1 LOOP2
     do
       echo $LOOP1
       kill -9 $LOOP1
     done <$HOLD1
  rm /tmp/*.$$
  ;;
N|n);;
esac
```

运行该脚本将会产生如下的输出：

```
$ pskill web
1760 ./webmon
1761 /usr/apps/web_col
Are these the processes to be killed ? [y..n] >y

1760
1761
[1]+  Killed                    webmon
```

在使用该脚本时，要确信存在相应的进程：

```
$ pskill web
No processes found..sorry
```

26.2.2 检测信号

有些信号可以被应用程序或脚本捕获，并依据该信号采取相应的行动。另外一些信号不能被捕获。例如，如果一个命令收到了信号9，就无法再捕捉其他信号。

在编写shell脚本时，只需关心信号1、2、3和15。当脚本捕捉到一个信号后，它可能会采取下面三种操作之一：

1) 不采取任何行动，由系统来进行处理。

2) 捕获该信号，但忽略它。

China-pub.com 下载

3) 捕获该信号，并采取相应的行动。

大多数的脚本都使用第一种处理方法，这也是到目前为止本书中所有脚本所采取的处理方法。

如果想要采取另外两种处理方法，必须使用 trap命令。

26.3 trap

trap可以使你在脚本中捕捉信号。该命令的一般形式为：

trap name signal(s)

其中，name是捕捉到信号以后所采取的一系列操作。实际生活中， name一般是一个专门用来处理所捕捉信号的函数。 Name需要用双引号（“ ”）引起来。 Signal就是待捕捉的信号。

脚本在捕捉到一个信号以后，通常会采取某些行动。最常见的行动包括：

1) 清除临时文件。

2) 忽略该信号。

3) 询问用户是否终止该脚本的运行。

下表列出了一些最常见的trap命令用法：

trap "" 2 3	忽略信号2和信号3，用户不能终止该脚本
trap"commands" 2 3	如果捕捉到信号2或3，就执行相应的commands命令
trap 2 3	复位信号2和3，用户可以终止该脚本

也可以使用单引号（‘ ’）来代替双引号（“ ”）；其结果是一样的。

26.3.1 捕获信号并采取相应的行动

下面的例子一经运行就开始计数直至用户按 <Ctrl-C>（信号2）。这时该脚本将会显示出当前的循环数字，然后退出。

在本例中trap命令的格式为：

trap "do_something" signal no:(s)

下面就是该脚本：

```
$ pg trap1
#!/bin/sh
#trap1
trap "my_exit" 2
LOOP=0
my_exit()
{
echo "You just hit <CTRL-C>, at number $LOOP"
echo " I will now exit "
exit 1
}

while :
do
  LOOP=`expr $LOOP + 1`
  echo $LOOP
done
```

现在让我们来仔细分析一下该脚本。

China-pub.com 下载


```
trap "my_exit" 2
```

在本例中，由于设置了trap命令，所以在捕捉到信号2以后，双引号内的my_exit函数将被执行。

```
my_exit()
{
echo "You just hit <CTRL-C>, at number $LOOP"
echo " I will now exit "
exit 1
}
```

函数my_exit将在脚本捕捉到信号2后被调用；用户将会看到$LOOP变量的内容，即用户按<Ctrl-C>时的计数值。在实际中，通常捕捉到信号2后所调用的函数是用来完成清除临时文件等任务的。

下面是该脚本的运行结果：

```
$ trap1
1
...
...
211
212
You just hit <CTRL-C>, at number 213
 I will now exit
```

26.3.2 捕获信号并采取行动的另一个例子

下面就是一个捕获信号后清除临时文件的例子。

下面的脚本在运行时不断使用df和ps命令向临时文件HOLD1.$$和HOLD2.$$中写入相应的信息。你应该还记得$$表示当前的进程号。当用户按<CTRL-C>时，这些临时文件将被清除。

```
$ pg trap2
#!/bin/sh
# trap2
# trap only signal 2....<CTRL-C>
trap "my_exit" 2
HOLD1=/tmp/HOLD1.$$
HOLD2=/tmp/HOLD2.$$

my_exit()
{
# my_exit
echo "<CTRL-C> detected..Now cleaning up..wait"
# delete the temp files
rm /tmp/*.$$ 2> /dev/null
exit 1
}

echo "processing...."
# loop forever, do some processing
while :
do
  df >>$HOLD1
```

China-pub.com 下载

```
  ps xa >>$HOLD2
done
```

上面的脚本在运行时会产生如下的结果：

```
$ trap2
processing....
<CTRL-C> detected..Now cleaning up..wait
```

当收到信号2或3时，尽管一般情况下这都不是误操作，但是为了安全起见，不妨给用户一个选择的机会，这样用户在不小心按下 <CTRL-C>后，仍然可以撤消刚才的动作。

在下面的例子中，在脚本捕捉到信号 2后将会向用户提供一个选择，询问用户是否真的要退出。这里使用case语句来决定采取何种操作。

如果用户希望退出，他或她可以选择 1，此时当前函数会以状态 1退出，而另一个清除进程将会据此启动。如果用户并不希望退出，那么可以选择 2或不做任何选择，此时 case语句将会使用户退回到脚本中原来的地方。在 case语句中一定要包含用户输入空字符串的情况。

下面的函数在收到信号后，将会向用户提供选择：

```
my_exit()
{
# my_exit
echo -e "\nReceived interrupt ... "
echo "Do you really wish to exit ???"
echo " 1: Yes"
echo " 2: No"
echo -n " Your choice [1..2] >"
read ANS
case $ANS in
1) # cleanup temp files.. etc..
  exit 1
  ;;
2) # do nothing
3) ;;
esac
}
```

下面是完整的脚本：

```
$ pg trap4
#!/bin/sh
# trap4
# trap signal 1 2 3 and 15
trap "my_exit" 1 2 3 15

LOOP=0

# temp files
HOLD1=/tmp/HOLD1.$$
HOLD2=/tmp/HOLD2.$$
my_exit()
{
# my_exit
echo -e "\nRecieved interrupt..."
echo "Do you wish to really exit ???"
echo " Y: Yes"
echo " N: No"
```

China-pub.com
下载


```
echo -n " Your choice [Y..N] >"
read ANS
case $ANS in
Y|y) exit 1;;     # exit the script
N|n) ;;           # return to normal processing
esac
}

# a while loop here perhaps for reading in fields
echo -n "Enter your name :"
read NAME
echo -n "Enter your age :"
read AGE
```

当上面的脚本运行时，只要在输入任何域时按下 <CTRL-C>，就会得到一个选择：是继续运行还是退出。

```
$ trap4
Enter your name :David Ta
Received interrupt...
Do you really wish to exit ???
 1: Yes
 2: No
 Your choice [1..2] >2

Enter your age :
```

26.3.3 锁住终端

下面的脚本是另一个捕获信号的例子。该脚本名为 lockit，它将使用一个连续不断的 while 循环锁住终端。在该脚本中， trap命令捕捉信号2、3和15。如果一个用户试图中断该脚本的运行，将会得到一个不成功的提示。

在脚本初次执行时，将会被提示输入一个口令。在解锁终端时没有任何提示，可以直接输入口令并按回车键。该脚本会从终端读入所输入的口令，并与预先设置的口令做比较，如果一致就解锁终端。

如果忘记了自己的口令，那么只好登录到另一个终端上并杀死该进程。在本例中没有对口令的长度加以限制——这完全取决于你。

如果你从另外一个终端上杀死了该进程，当再次回到这个终端时，可能会遇到终端设置问题，例如回车键不起作用。这时可以试着使用下面的命令，这样可以解决大部分问题。

```
$ stty sane
```

下面就是该脚本。

```
$ pg lockit
#!/bin/sh

# lockit
# trap signals 2 3 and 15
trap "nice_try" 2 3 15

# get the device we are running on
TTY=`tty`
```

China-pub.com 下载

```
nice_try()
{
# nice_try
echo "Nice try, the terminal stays locked"
}

# save stty settings hide characters typed in for the password
SAVEDSTTY=`stty -g`
stty -echo
echo -n "Enter your password to lock $TTY :"
read PASSWORD
clear

while :
do
  # read from tty only !!,
  read RESPONSE < $TTY
  if [ "$RESPONSE" = "$PASSWORD" ]; then
    # password matches...unlocking
    echo "unlocking..."
    break
  fi

  # show this if the user inputs a wrong password
  # or hits return

  echo "wrong password and terminal is locked.."
done

# restore stty settings
stty $SAVEDSTTY
```

下面是lockit脚本运行时的输出：

```
$ lockit
Enter your password to lock /dev/ttyS1 :
```

接着屏幕就被清除。如果按回车键或其他错误的口令，该脚本将会输出：

```
wrong password and terminal is locked..
Nice try, the terminal stays locked
wrong password and terminal is locked..
Nice try, the terminal stays locked
wrong password and terminal is locked..
```

现在输入正确的口令：

```
unlocking...
$
```

现在又回到命令提示符下了。

26.3.4 忽略信号

在用户登录时，系统将会执行/etc/profile文件，根用户不希望其他普通用户打断这一进程。他通常通过设置trap来屏蔽信号1、2、3和15，然后在用户读当天的消息时重新打开这些信号。最后仍然回到屏蔽这些信号的状态。

在编写脚本时也可以采用类似的办法。在脚本运行的某些关键时刻，比如打开了很多文

China-pub.com
下载

件时，不希望该脚本被中断，以免破坏这些文件。通过设置 trap来屏蔽某些信号就可以解决这个问题。在这些关键性的处理过程结束后，再重新打开信号。

忽略信号的一般格式为（信号9除外）：

```
trap""signal no:(s)
```

注意，在双引号之间没有任何字符，为了重新回到捕捉信号的状态，可以使用如下的命令：

```
trap"do something" signal no:(s)
```

下面我们来总结一下上述方法。

trap ""1 2 3 15：忽略信号。

关键性的处理过程

trap"my_exit" 1 2 3 15：重新回到捕捉信号的状态，在捕捉到信号后调用 my_exit函数。

下面就是一个这样的例子，其中的“关键”过程实际上是一个 while循环，但它能够很好地说明这种方法。在第一个循环中，通过设置 trap来屏蔽信号，但是在第二个例子中，又回到捕捉信号的状态。

两个循环都只数到6，不过在循环中使用了一个 sleep命令，这样就可以有充分的时间来实验中断该循环。

下面就是脚本。

```
$ pg trap_ignore
#!/bin/sh
# trap_ignore
# ignore the signals
trap "" 1 2 3 15

LOOP=0
my_exit()
# my_exit
{
echo "Received interrupt on count $LOOP"
echo "Now exiting..."
exit 1
}

# critical processing, cannot be interrupted....
LOOP=0
while :
do
  LOOP=`expr $LOOP + 1`
  echo "critical processing..$LOOP..you cannot interrupt me"
  sleep 1
  if [ "$LOOP" -eq 6 ]; then
    break
  fi
done

LOOP=0
# critical processing finished, now set trap again but this time allow
interrupts.
trap "my_exit" 1 2 3 15
```

```
while :
do
  LOOP=`expr $LOOP + 1`

  echo "Non-critical processing..$LOOP..interrupt me now if you want"
  sleep 1
  if [ "$LOOP" -eq 6 ]; then
    break
  fi
done
```

在上面的脚本在运行时，如果我们在第一个循环期间按下 <Ctrl-C>，它不会有任何反应，这是因为我们通过设置trap屏蔽了信号；而在第二个循环中由于重新回到捕捉信号的状态，按下<Ctrl-C>就会调用my_exit函数。

```
$ trap_ignore
critical processing..1..you cannot interrupt me
critical processing..2..you cannot interrupt me
critical processing..3..you cannot interrupt me
critical processing..4..you cannot interrupt me
critical processing..5..you cannot interrupt me
critical processing..6..you cannot interrupt me
Non-critical processing..1..interrupt me now if you want
Non-critical processing..2..interrupt me now if you want
Received interrupt on count 2
Now exiting...
```

当脚本捕获到信号时，通过使用trap命令，可以更好地控制脚本的运行。捕获信号并进行处理是一个脚本健壮性的标志。

26.4 eval

eval命令将会首先扫描命令行进行所有的置换，然后再执行该命令。该命令适用于那些一次扫描无法实现其功能的变量。该命令对变量进行两次扫描。这些需要进行两次扫描的变量有时被称为复杂变量。不过我觉得这些变量本身并不复杂。

eval命令也可以用于回显简单变量，不一定是复杂变量。

```
$ NAME=Honeysuckle
$ eval echo $NAME
Honeysuckle
$ echo $NAME
Honeysuckle
```

解释eval命令是怎么回事的最好办法就是看几个例子。

26.4.1 执行含有字符串的命令

我们首先创建一个名为testf的小文件，在这个小文件中含有一些文本。接着，将 cat testf赋给变量MYFILE，现在我们echo该变量，看看是否能够执行上述命令。

```
$ pg testf
May Day, May Day
Going Down
```

现在我们将cat testf赋给变量MYFILE。

```
$ MYFILE="cat testf"
```

如果我们echo该变量，我们将无法列出testf文件中的内容。

```
$ echo $MYFILE
cat testf
```

让我们来试一下eval命令，记住eval命令将会对该变量进行两次扫瞄。

```
$ eval $MYFILE
May Day, May Day
Going Down
```

从上面的结果可以看出，使用eval命令不但可以置换该变量，还能够执行相应的命令。第一次扫描进行了变量置换，第二次扫描执行了该字符串中所包含的命令 cat testf。

下面是另一个例子。一个名为CAT_PASSWD的变量含有字符串“ cat /etc/passwd | more ”。eval 命令可以执行该字符串所对应的命令。

```
$ CAT_PASSWD="cat /etc/passwd | more"
$ echo $CAT_PASSWD
cat /etc/passwd|more
$ eval $CAT_PASSWD
root:HccPbzT5tb00g:0:0:root:/root:/bin/sh
bin:*:1:1:bin:/bin:
daemon:*:2:2:daemon:/sbin:
adm:*:3:4:adm:/var/adm:
...
...
```

eval命令还可以用来显示出传递给脚本的最后一个参数。现在来看下面的这个例子。

```
$ pg evalit
#!/bin/sh
# evalit
echo " Total number of arguments passed is $#"
echo " The process ID is $$"
echo " Last argument is " $(eval echo \$$#)
```

在运行上述脚本时，我们会看到如下的结果（你所看到进程号可能会不一样）：

```
$ evalit alpha bravo charlie
 Total number of arguments passed is 3
 The process ID is 780
 Last argument is  charlie
```

在上面的脚本中， eval命令首先把$$#解析为当前shell的参数个数，然后在第二次扫描时得出最后一个参数。

26.4.2 给每个值一个变量名

可以给一个值一个变量名。下面我对此做些解释，假定有一个名为 data的文件：

```
$ pg data
PC      486
MONITOR svga
NETWORK yes
```

你希望该文件中的第一列成为变量名，第二列成为该变量的值，这样就可以：

```
echo $PC
486
```

怎样才能做到这一点呢？当然是使用eval命令。

China-pub.com 下载

```
$ pg eval_it
#!/bin/sh
#eval_it
while read NAME TYPE
do
  eval `echo "${NAME}=${TYPE}"`
done < data
echo "You have a $PC pc, with a $MONITOR monitor"
echo "and are you network ? $NETWORK"
```

我们用data文件的第一行来解释上述脚本的执行过程，该脚本读入“ PC ”和“ 486 ”两个词，把它们分别赋给变量 NAME和TYPE。Eval命令的第一次扫描把 NAME和TYPE分别置换为“ PC ”和“ 486 ”，第二次扫描时将PC作为变量，并将“ 486 ”作为变量的值。

下面是运行上述脚本的结果：

```
$ eval_it
You have a 486 pc, with a svga monitor
and are you network ? yes
```

eval命令并不是一个在脚本中很常见的命令，但是如果需要对变量进行两次扫瞄的话，就要使用eval命令了。

26.5 logger命令

系统中含有相当多的日志文件。其中的一个日志文件叫作 messages，它通常位于/var/adm或/var/log目录下。一个名为syslog的配置文件可以用来定义记录在messages文件中的消息，这些消息有一定的格式。如果想知道系统中的相应配置，可以查看 /etc/syslog.conf文件。该文件中包含了用于发送各种不同类型消息的工具及它们的优先级。

这里我们并不想深入探讨UNIX和LINUX是如何向该文件中记录信息的。我们现在只要知道这些消息有不同的级别，从信息性的消息到关键性的消息。

还可以使用logger命令向该文件发送消息。在使用该命令之前，最好查阅连机手册，因为在不同供应商所提供的操作系统上该命令的语法也有所不同。

不过，由于这里只涉及到信息性的消息，因此不必担心下面的命令不安全。

你可能会出于下列的原因向该文件中发送消息：

- 在某一个特定的时间段出现的访问或登录。
- 你的某些执行关键任务的脚本运行失败。
- 监控脚本的报告。

下面是/var/adm/messages文件的例子。在系统上所看到的相应文件可能和下面的例子有少许差别。

```
$ tail /var/adm.messages
Jun 16 20:59:03 localhost login[281]: DIALUP AT ttyS1 BY root
Jun 16 20:59:03 localhost login[281]: ROOT LOGIN ON ttyS1
Jun 16 20:59:04 localhost PAM_pwdb[281]: (login) session closed for user
root
Jun 16 21:58:38 localhost named[211]: Cleaned cache of 0 RRs
Jun 16 21:58:39 localhost named[211]: USAGE 929570318 929566719
Jun 16 21:58:39 localhost named[211]: NSTATS 929570318 929566719
```

logger命令的一般形式为：

```
logger -p -I message
```

其中：

-p：为优先级，这里只涉及到提示用户注意的优先级，这也是缺省值。

-i：在每个消息中记录发送消息的进程号。

26.5.1 使用logger命令

可以使用如下命令：

```
$ logger -p notice "This is a test message.Please Ignore $LOGNAME"
```

可能需要等几分钟才能看到该消息被记录到 message文件中。

```
$ tail /var/adm/messages
...
...
Jun 17 10:36:49 acers6 dave: This is a test message.Please Ignore dave
```

如你所见，发送这一消息的用户也被记录了下来。

现在来创建一个小小的脚本，用它来记录当前系统中的用户数。该脚本可以在一天的时段中记录系统的使用率。只要把它放进 crontab文件中，使它每30分钟运行一次即可。

```
$ pg test_logger
#!/bin/sh
# test_logger
logger -p notice  "`basename $0`:there are currently `who |wc -l` users on
                   the system"
```

运行下面的脚本。

```
$ test_logger
```

现在来看看message文件的末尾：

```
$ tail /var/adm/messages
...
...
Jun 17 11:02:53 acers6 dave: test_script:there are currently 15 users on
the system
```

26.5.2 在脚本中使用logger命令

向日志文件中发送信息的一个更为合理的用途就是用于脚本非正常退出时。如果希望向日志文件中发送消息，只要在捕获信号的退出函数中包含 logger命令即可。

在下面的清除脚本中，如果该脚本捕获到信号 2、3或15的话，就向该日志文件发送一个消息。

```
$ pg cleanup
# !/bin/sh
# cleanup
# cleanup system logs
trap "my_exit" 2 3 15

my_exit()
{
# my_exit
logger -p notice " `basename $0`: Was killed whilst cleaning up system
                  logs..CHECK OUT ANY DAMAGE"
```

China-pub.com 下载

```
exit 1
}

tail -3200c /var/adm/utmp > /tmp/utmp
mv /tmp/utmp /var/adm/utmp
>/var/adm/wtmp
#
tail -10 /var/adm/sulog > /tmp/o_sulog
mv /tmp/o_sulog /var/adm/sulog
...
```

这样只要看一下这个日志文件就可以知道脚本的运行结果是否正常。

```
$ tail /var/adm/messages
...
...
Jun 17 11:34:28 acers6 dave: cleanup:Was killed whilst cleaning up
systemlogs..       CHECK OUT ANY DAMAGE
```

除了使用logger命令对一些关键性的脚本处理过程做日志外，我还用它来记录使用调制解调器连接系统的用户。下面的一段脚本记录了从串口 tty0和tty02连接到系统中的用户。这部分代码来自于我编写的一个/etc/profile文件。

```
TTY_LINE=`tty`
case $TTY_LINE in
"/dev/tty0")
  TERM=ibm3151
  ;;
"/dev/tty2")
  TERM=vt220
  # checks for allowed users on the modem line
  #
  echo "This is a modem connection"
  # modemf contains login names of valid users
  modemf=/usr/local/etc/modem.users
  if [-s $modemf ]
  then
    user=`cat $modemf| awk '{print $1}' | grep $LOGNAME`
    # if your name is not in the file, you are not coming in
    if [ "$user" != "$LOGNAME" ]
    then
      echo "INVALID USER FOR MODEM CONNECTION"
      echo " DISCONNECTING,,,,,,"
      sleep 1
      exit 1
    else
      echo "modem connection allowed"
    fi
  fi
  logger -p notice "modem line connect $TTY_LINE..$LOGNAME"
  ;;
*) TERM=vt220
  stty erase '^h'
  ;;
esac
```

China-pub.com
下载

当希望在系统全局的日志文件中记录信息的时候， logger命令是一个非常好的工具。

26.6 小结

理解信号和对信号的捕获可以使脚本的退出更为完整。通过在系统日志文件中记录信息，你或系统管理员就能够更容易地发现存在的问题。

China-pub.com 下载

第27章 几个脚本例子

本章包含了我最常用的几个脚本。你会发现它们都相当短小而简单。这就是脚本的一个优点；它不是很长、很复杂，只需很短的代码就能够完成相当多的功能，可以节约大量的时间。

本章中包含以下内容：

• 各种脚本的例子。

我本来打算在本章中提供一个通用的数据验证数据库脚本，但是由于它超过了 500行，我觉得编辑肯定不会同意把它收入书中。那个脚本几年前只有几行，后来由于不断增加功能，变成了现在这么长。最后，我选择了如下六个脚本作为例子：

pingall：一个按照/etc/hosts文件中的条目逐一ping所有主机的脚本。

backup_gen：一个通用的备份脚本，能够加载缺省设置。

del.lines：一个引用sed命令的脚本，能从文件中删除若干行。

access_deny：一个能够阻止某些特定用户登录的工具。

logroll：一个能够清除超过某一长度的日志的工具。

nfsdown：一个快速unmount所有nfs文件系统的工具。

27.1 pingall

几年前我写了一个名为 pingall的脚本在夜间运行，把它作为常规报告脚本的一部分。它能够按照/etc/hosts文件中的条目逐一ping所有的主机。

该脚本列出/etc/hosts文件并查找其中的非注释行（不以 #开头的行）。然后使用一个 while循环读入所有的行，接下来使用 awk分析出每行的第一个域，并把它赋给变量 ADDR。最后使用for循环逐一ping相应的地址。

下面就是该脚本。

```
$ pg pingall
#!/bin/sh
# pingall

# grab /etc/hosts and ping each address
cat /etc/hosts| grep -v '^#' | while read LINE
do
  ADDR=`awk '{print $1}'`
  for MACHINE in $ADDR
  do
    ping -s -c1 $MACHINE
  done
done
```

上述脚本可以很容易地进行扩展，加进其他网络报告工具。

27.2 backup_gen

在本章中我选择了这个脚本并不是因为它展示了如何备份目录，而是因为它是一个同其

China-pub.com 下载

他脚本共享设置的很好例子。

backup_gen是一个用于备份的脚本，它从一个缺省的配置文件中读入设置，然后根据这些参数对系统进行备份。用户可以根据自己的需要改变这些缺省设置。这是一个不同脚本如何使用相同设置或仅在自己运行期间改变相应设置的极好例子。当该脚本执行时，它首先确认源文件backup.defaults是否存在，如果不存在，则退出。

该脚本在运行时，会显示出一个题头和缺省设置，并询问用户是否需要改变任何缺省设置。如果用户回答“是”，在他们修改设置之前，该脚本就会提示他们输入一个代码，用户可以有三次机会；如果输入正确的代码后仍无法改变设置，这就意味着用户必须要使用缺省设置。一般来说，在输入正确代码后，用户可以改变下列设置（[ ]中的为缺省设置）：

- 磁带设备[rmt0]　　可以选择rmt1和rmt3
- 备份完成后是否向系统管理员发邮件[是]　　可以选择否
- 备份的类型[全备份]　　可以选择普通备份或sybase备份

脚本中使用了一些临时变量来保存被修改的设置。用户可以按回车键选择缺省设置。下列设置不能被改变：

备份日志文件名。

用户代码。

接着所有的改变会生效。在这些改变生效之后，相应的临时变量又会被重新赋予缺省值。在备份进行之前，首先要测试磁带设备。备份过程使用 find和cpio命令，它们从设置文件中读入相应变量的缺省值，或使用用户设定的值。

下面就是该脚本。

```
$ pg backup_run
#!/bin/sh
# backup_run

# script to run the backups
# loads in a setting file for the user to change

SOURCE=/appdva/bin/backup.defaults
check_source()
{
# check_source
# can we load the file
# backup.defaults is the source file containing config/functions
# make sure your path includes this directory you are running from
if [ -r $SOURCE ]; then
  . /$SOURCE
else
  echo "`basename $0`: cannot locate defaults file"
  exit 1
fi
}

header()
{
# header
USER=`whoami`
MYDATE=`date +%A" "%e" of "%B-%Y`
```

China-pub.com 下载

```
clear
cat << MAYDAY
User : $USER                                              $MYDATE
                          NETWORK SYSTEM BACKUP
                          =====================
MAYDAY
}

change_settings()
{
# change_settings
# let the user see the default settings..
header
echo "Valid Entries Are..."
echo "Tape Device: rmt0, rmt1, rmt3"
echo "Mail Admin: yes, no"
echo "Backup Type: full, normal, sybase "
while :
do
  echo -n -c "\n\n Tape Device To Be Used For This Backup  [$_DEVICE] :"
  read T_DEVICE
  : ${T_DEVICE:=$_DEVICE}
  case $T_DEVICE in
  rmt0|rmt1|rmt3) break;;
  *) echo "The devices are either ... rmt0, rmt1, rmt3"
    ;;
  esac
done

# if the user hits return on any of the fields, the default value will be
used
while :
do
  echo -n " Mail Admin When Done                    [$_INFORM] :"
  read T_INFORM
  : ${T_INFORM:=$_INFORM}
  case $T_INFORM in
  yes|Yes) break;;
  no|No) break;;
  *) echo "The choices are yes, no"
    ;;
  esac
done

while :
do
  echo -n " Backup Type                              [$_TYPE] :"
  read T_TYPE
  : ${T_TYPE:=$_TYPE}
  case $T_TYPE in
  Full|full) break;;
  Normal|normal)break;;
  Sybase|sybase)break;;
  *) echo "The choices are either ... full, normal, sybase"
```

China-pub.com
下载


```
  esac
done
# re-assign the temp variables back to original variables that
# were loaded in
_DEVICE=$T_DEVICE; _INFORM=$T_INFORM; _INFORM=$T_INFORM
}

show_settings()
# display current settings
{
cat << MAYDAY
                        Default Settings Are...
                     Tape Device To Be Used        : $_DEVICE
                     Mail Admin When Done          : $_INFORM
                     Type Of Backup                : $_TYPE
                     Log file of backup            : $_LOGFILE
MAYDAY
}

get_code()
{
# users get 3 attempts at entering the correct code
# _CODE is loaded in from the source file
clear
header
_COUNTER=0
echo " YOU MUST ENTER THE CORRECT CODE TO BE ABLE TO CHANGE
     DEFAULT SETTINGS"
while :
do
  _COUNTER=`expr $_COUNTER + 1`
  echo -n "Enter the code to change the settings  :"
  read T_CODE
  # echo $_COUNTER
  if [ "$T_CODE" = "$_CODE" ]; then
    return 0
  else
    if [ "$_COUNTER" -gt 3 ]; then
      echo "Sorry incorrect code entered, you cannot change the settings.."
      return 1
    fi
  fi
done
}

check_drive()
{
# make sure we can rewind the tape
mt -f /dev/$_DEVICE rewind > /dev/null 2>&1
if [ $? -ne 0 ]; then
  return 1
else
  return 0
fi
```

China-pub.com 下载

```
}
#========== main==============

# can we source the file
check_source
header
# display the loaded in variables
show_settings
# ask user if he/she wants to change any settings
if continue_prompt "Do you wish To Change Some Of The System Defaults" "Y";
then
  # yes then enter code name
  if get_code; then
    # change some settings
    change_settings
  fi
fi

#------- got settings.. now do backup
if check_drive; then
  echo "tape OK...."
else
  echo "Cannot rewind the tape..Is it in the tape drive ???"
  echo "Check it out"
  exit 1
fi

# file system paths to backup
case $_TYPE in
Full|full)
  BACKUP_PATH="sybase syb/support etc var bin apps use/local"
  ;;
Normal|normal)
  BACKUP_PATH="etc var bin apps usr/local"
  ;;
Sybase|sybase)
  BACKUP_PATH="sybase syb/support"
  ;;
esac
# now for backup
cd /
echo "Now starting backup......"
find $BACKUP_PATH -print | cpio -ovB -O /dev/$_DEVICE >> $_LOGFILE 2>&1

# if the above cpio does not work on your system try cpio below, instead
# find $BACKUP_PATH -print | cpio -ovB > /dev/$_DEVICE >> $_LOGFILE 2>&1

# to get more information on the tape change -ovB to -ovcC66536

if [ "$_INFORM" = "yes" ]; then
  echo "Backup finished check the log file" | mail admin
fi
```

源文件backup.defaults中包含函数continue_prompt，还有所有缺省设置。下面就是该源文件。

```
$ pg backup.defaults
#!/bin/sh
# backup.defaults
# configuration default file for network backups
# edit this file at your own risk !!
# name backup.defaults
#---------------------------------
# not necessary for the environments to be in quotes..but hey!
_CODE="comet"
_LOGFILE="/appdva/backup/log.`date +%y%m%d`"
_DEVICE="rmt0"
_INFORM="yes"
_TYPE="Full"
#----------------------------------
continue_prompt()
# continue_prompt
# to call: continue_prompt "string to display" default_answer
{
_STR=$1
_DEFAULT=$2
# check we have the right params
if [ $# -lt 1 ]; then
  echo "continue_prompt: I need a string to display"
  return 1
fi
while :
do
  echo -n "$_STR [Y..N] [$_DEFAULT]:"
  read _ANS
  : ${_ANS:=$_DEFAULT}
    if [ "$_ANS" = "" ]; then
      case $_ANS in
      Y) return 0 ;;
      N) return 1 ;;
      esac
    fi
  # user has selected something
  case $_ANS in
  y|Y|Yes|YES)
    return 0
    ;;
  n|N|No|NO)
    return 1
    ;;
  *) echo "Answer either Y or N, default is $_DEFAULT"
    ;;
  esac
  echo $_ANS
done
}
```

下面是该脚本运行时的输出，缺省设置被显示在屏幕上，用户被询问是否要改变这些设置：

```
User : dave                                           Tuesday 15 of June-1999
                        NETWORK SYSTEM BACKUP
```

China-pub.com 下载

```
                         ========================
                         Default Settings Are...
               Tape Device To Be Used      : rmt0
               Mail Admin When Done        : yes
               Type Of Backup              : Full
               Log file of backup          : /appdva/backup/log.990615
      Do you wish To Change Some Of The System Defaults [Y..N] [Y]:
```

下面是用户改变缺省值的过程。在下面的例子中，备份类型被用户改变，但是该脚本在检查了相应的磁带设备之后，发现它有点问题。在使用了最后一个状态命令之后，该脚本将会退出。

```
User : dave                                          Tuesday 15 of June-1999
                        NETWORK SYSTEM BACKUP
                        ========================
Valid Entries Are...
Tape Device: rmt0, rmt1, rmt3
Mail Admin: yes, no
Backup Type: full, normal, sybase

 Tape Device To Be Used For This Backup  [rmt0] :
 Mail Admin When Done                     [yes] :
 Backup Type                              [Full : Normal
Cannot rewind the tape..Is it in the tape drive ???
Check it out
```

27.3 del.lines

之所以要编写这个脚本，是因为应用程序开发者总是问我“用sed的哪个命令删除空行？”。我决定写一个小脚本给他们使用，以免他们老是打电话问我这个命令。

这个脚本只是包装了一下sed命令，但它能够使用户很方便地使用，他们非常喜欢用。

脚本一般都不长。如果你认为写一个脚本能够使某些任务自动化，能够节约时间，那么你就可以编写一个脚本。

这个脚本可以处理一个或多个文件。每个文件在用 sed删除空行之前要先核实是否存在。sed的输出被导入一个文件名中含有 $$的临时文件，最后这个临时文件又被移回到原来的文件中。

该脚本使用shift命令取得所有的文件名，用 while循环逐个处理所有的文件，直至处理完为止。

可以使用del.lines -help获得一个简短的帮助。你也可以创建一个更好的帮助。

下面是该脚本。

```
$ pg del.lines
#!/bin/sh

# del.lines
# script takes filename(s) and deletes all blank lines

TEMP_F=/tmp/del.lines.$$

usage()
{
```

China-pub.com
下载


```
# usage
echo "Usage :`basename $0` file [file..]"
echo "try `basename $0` -help for more info"
exit 1
}

if [ $# -eq 0 ]; then
  usage
fi

FILES=$1
while [ $# -gt 0 ]
do
  echo "..$1"
  case $1 in
  -help)  cat << MAYDAY
    Use this script to delete all blank lines from a text file(s)
    MAYDAY
    exit 0
    ;;
  *) FILE_NAME=$1

    if [ -f $1 ]; then
      sed '/^$/d' $FILE_NAME >$TEMP_F
      mv $TEMP_F $FILE_NAME

    else
      echo "`basename $0` cannot find this file : $1"
    fi
    shift
    ;;
   esac
 done
```

27.4 access.deny

在对系统进行某些更新时，你可能不希望用户登录，这时可以使用 /etc/nologin文件，大多数系统都提供这个文件。一旦在 /etc目录中使用touch命令创建了一个名为nologin的文件，除root以外的任何用户都将无法登录。

如果系统不支持这种方法，你一样还可以做到这点——可以自己创建这个文件，下面就是具体的做法。

可以在/etc/profile文件中加入下面的代码：

```
if [ -f /etc/nologin ]; then
  if [ $LOGNAME != "root" ]; then
    echo "Sorry $LOGNAME the system is unavailable at the moment"
    exit 1
  fi
fi
```

现在，可以通过在/etc目录下创建nologin文件来阻止除根用户以外的其他用户登录。记住，该文件要对所有用户可读。

China-pub.com 下载

```
touch /etc/nologin
chmod 644 /etc/nologin
```

当决定恢复用户登录时，只要删除该文件即可。

```
rm /etc/nologin
```

上述办法可以很方便地组织除根用户外的所有用户登录。如果希望临时禁止某个用户登录，可以修改/etc/passwd文件，把该用户的口令域的第一个字符变成 *。不过，这个问题比较复杂，在操作之前一定要搞清楚，否则会带来系统性的问题。

LINUX提供了一个工具，可以通过它在 login.access文件中写入用户名和用户组。该文件可以用来允许或禁止用户对系统的访问。

这里有一个上述工具的简化版本 deny.access。该脚本从/etc/profile文件中运行，它读入一个名为lockout.users的文件。该文件包含有禁止登录的用户名。如果该文件中出现了 all这个单词，那么除root以外的所有用户都将被禁止登录。

下面是lockout.users文件的一个例子，该文件可以包含注释行。

```
$ pg lockout.users
# lockout.users
# put the user names in this file, that you want
# locked out of the system.
# Remove the user names from this file, to let the users back in.
# peter is on long holiday back next month
peter
# lulu is off for two weeks, back at the end of the month
lulu
#dave
#pauline
```

下面解释该脚本的工作过程。首先，通过设置trap忽略所有的信号，这样用户就无法中断它的执行。如果文件lockout.users存在，那么脚本将会继续运行。它首先检查该文件中是否存在单词all，如果存在，就不再检查该文件中的其他用户名，并禁止除根用户以外的所有其他用户登录。不要使用注释来屏蔽单词all，因为这样它仍然有可能起作用。不过你可以注释用户名。

如果单词all被找到，那么除root外的所有用户都将无法登录。为了准确起见，在该脚本中使用了grep的精确匹配模式all\>。这时用户将会在屏幕上看到系统不可用的消息。

该脚本中的主要函数是get_users 。它读入文件lockout.users，忽略所有以#开头的注释行。它通过比较用户名来确保用户名 root没有出现在该文件中，即使出现也不会禁止 root登录，否则后果将难以想象。

当前正在登录的用户名可以从变量 LOGNAME中得到，并与变量NAMES做比较，而变量NAMES的内容来自于lockout.users文件。如果匹配，相应用户的登录进程将被终止。

我在几个拥有近40个用户的系统上运行该脚本，它并没有影响用户登录的速度。当用户外出超过一周或者用户午餐，而我需要对系统进行更新时，我就使用该脚本临时锁住相应的帐户。

需要在/etc/profile文件中加入这样一行。我把它加在该文件的末尾，这样即使用户无法登录，也可以在此之前看见当前发给他的新消息。

```
. /apps/bin/deny.access
```

/apps/bin目录是我存放全局性脚本的地方——你可能把这些脚本放在另外的目录中，不过

一定要确保所有用户都对该脚本及存放它的目录具有执行权限。

如果得到“权限不足”的错误提示，那说明该脚本或目录的权限不足。

我的lockout.users文件放在/apps/etc目录下。如果你的系统的目录结构有所不同的话，应该作出相应的调整。由于该文件在登录时被引用，可以使用 set命令看到相应的函数（不过无法看到lockout.users文件）。如果你觉得这不妥，只要在这些函数执行后使用 unset命令去掉它们即可。可以把unset命令直接放在/etc/profile文件中该命令行之后，就像这样：

```
unset getusers
```

下面就是该脚本。

```
$ pg deny.access
#!/bin/sh
# deny.access

trap "" 2 3
# CHANGE BELOW IF YOU CHANGE THE LOCATION OF LOCKOUT.USERS
LOCKOUT=/apps/etc/lockout.users
MSG="Sorry $LOGNAME, your account has been disabled, ring the
administrator"
MSG2="Sorry $LOGNAME, the system is unavailable at the moment"

check_lockout()
# check_logout
# make sure we have a file containing names to lockout
{
if [ -r $LOCKOUT ] ; then
  return 0
else
  return 1
fi
}

get_users()
# get_users
# read the file, if their LOGNAME matches a name in lockout.users'
# then kick them out!
{
while read NAMES
do
  case $NAMES in
  \#*);;   #do ignore comments
  *)
  # just in case somebody tries to lockout root..not in this script they
  # won't
    if [ "$NAMES" = "root" ]; then
      break
    fi
    if [ "$NAMES" = "$LOGNAME" ]; then
    # let use see message before kicking them out
      echo $MSG
      sleep 2
      exit 1
    else
      # no match next iteration of reading the file
```

China-pub.com 下载

```
      continue
    fi
    ;;
  esac
done < $LOCKOUT
}
if check_lockout; then
  if grep 'all\>' $LOCKOUT >/dev/null 2>&1
then
  # first check that 'all' is not present. If it is then
  # keep everybody out apart from root
  if [ "$LOGNAME" != "root" ]; then
    echo $MSG2
    sleep 2
    exit 2
    fi
  fi
  # do normal users, if any
  get_users
fi
```

27.5 logroll

我的系统中的有些日志文件增长十分迅速，每天手工检查这些日志文件的长度并倒换这些日志文件（通常是给文件名加个时间戳）是非常乏味的。于是我决定编写一个脚本来自动完成这项工作。该脚本将提交给 cron进程来运行，如果某个日志文件超过了特定的长度，那么它的内容将被倒换到另一个文件中，并清除原有文件中的内容。

你可以很容易地改编这个脚本用于清除其他的日志文件。我使用另外一个脚本来清除我的系统日志文件，它每周运行一次，截断相应的日志文件。如果我需要再回头看这些日志文件，只需在备份中寻找即可，这些日志文件的备份周期为16周，这个周期长度应该说是足够了。

该脚本中日志文件的长度限制是由变量 BLOCK_LIMIT设定的。这一数字代表了块数目，在本例中是8块（每块大小为4K字节）。可以按照自己的需求把这一数字设得更高。所有我要检查的日志文件名都保存在变量 LOGS中。

这里使用了一个 for循环来依次检查每一个日志文件，使用 du命令来获取日志文件长度。如果相应的文件长度大于 BLOCK_LIMIT变量所规定的值，那么该文件将被拷贝到一个文件名含有时间戳的文件中，并改变这个文件所属的组，原先的文件长度将被截断为 0。

该脚本由cron每周运行几次，生成了一些文件名中含有时间戳的日志文件备份，这样如果系统出现了任何问题，我还可以回到这些备份中查找。

```
$ pg logroll
#!/bin/sh
# logroll
# roll over the log files if sizes have reached the MARK
# could also be used for mail boxes ?
# limit size of log
# 4096 k
BLOCK_LIMIT=8

MYDATE=`date +%d%m`
# list of logs to check...yours will be different!
```

China-pub.com 下载


```
LOGS="/var/spool/audlog /var/spool/networks/netlog /etc/dns/named_log"
for LOG_FILE in $LOGS
do
  if [ -f $LOG_FILE ] ; then
    # get block size
    F_SIZE=`du -a $LOG_FILE | cut -f1`
  else
    echo "`basename $0` cannot find $LOG_FILE" >&2
    # could exit here, but I want to make sure we hit all
    # logs
    continue
  fi

  if [ "$F_SIZE" -gt "$BLOCK_LIMIT" ]; then
    # copy the log across and append a ddmm  date on it
    cp $LOG_FILE $LOG_FILE$MYDATE
    # create / zero the new log
    >$LOG_FILE
    chgrp admin $LOG_FILE$MYDATE
  fi
done
```

27.6 nfsdown

如果系统中包含nfs文件系统，你将发现下面的脚本非常实用。我管理着几台主机，不时地需要在工作时间重启动其中的某台机器。这种重启动过程当然是越快越好。

由于我在好几个机器上都挂接了远程目录，我不想依靠系统的重启动过程来卸载这些 nfs文件系统，宁愿自己来完成这个工作。这样还可以更快一些。

只要运行这个脚本就可以迅速卸载所有的nfs文件系统，这样就能更快的重新启动机器。

该脚本的LIST变量中含有提供nfs目录的主机名。使用 for循环逐一卸载相应的目录，用grep命令在df命令的结果中查找nfs文件系统。nfs目录的mount形式为：

machine: remote_directory

这一字符串被保存在变量NFS_MACHINE中。在umount命令中使用了该变量。

下面就是该脚本：

```
$ pg nfsdown
#!/bin/sh
# nfsdown
LIST="methalpha accounts warehouse dwaggs"
for LOOP in $LIST
do
  NFS_MACHINE=`df -k | grep $LOOP | awk '{print $1}'`
  if [ "$NFS_MACHINE" != "" ]; then
    umount $LOOP
  fi
done
```

27.7 小结

本章中所提供的脚本都是我最常用的。正如前面所提到的，脚本不一定很长、很复杂，但是它却不失为一种高效的方法。

China-pub.com
下载

第28章　运行级别脚本

如果希望在系统启动时自动运行某些应用程序、服务或脚本，或者在系统重启动时能够正确地关闭这些程序，那么需要创建运行级别脚本。除一种 LINUX变体外，所有的LINUX版本都含有这种基于系统V的运行级别配置目录，就像其他UNIX版本那样。

既然所有的系统都含有这种类型的配置，我们在本章中将会对它加以介绍，但如果你的系统不含有这种目录，也不要紧。还可以通过其他方法在系统启动时自动运行程序；本章的后半部分也将介绍这些方法。

本章包含下列内容：

• 运行级别。
• 如何创建rc.scripts。
• 如何在不同的运行级别实现相应的rc.scripts。
• 如何从inittab中启动应用程序。

能够创建运行级别脚本就意味着能够更灵活地控制系统。如果需要在某一个特定的运行级别启动或停止程序，就得创建运行级别脚本（它们通常被称为rc.script）。

任何用关键字 start或stop调用的、能够启动或停止程序运行的脚本都可以看作是一个rc.script。注意，应当由用户来保证他或她所提交的脚本是一个有效的脚本，能够正确地启动或停止某一服务。

运行级别配置目录的机制使得rc.script只在系统切换运行级别时有效。它不负责检查某一运行级别中所有的特定服务是否都已经被启动或停止。这是 shell编程者的事。

还可以按照希望运行的服务来控制系统的运行级别，但是这已经超出了本书的讨论范围。

28.1　怎么知道系统中是否含有运行级别目录

rc.scripts一般保存在（实际上是个链接，这一点我们将在后面讲述）/etc/rcN.d或/etc/rc.d/rcN.d目录下，

其中，N是一个数字。通常是7个，因为rcN.d目录的序号是从0到6，不过在系统上可能会有另外几个目录，如rcS.d。这并不重要，这里我们只关心带有数字的目录。

```
$ pwd
/etc
drwxr-xr-x   2 root     sys          1024 Dec 22  1996 rc0.d
drwxr-xr-x   2 root     sys          1024 Dec 22  1996 rc1.d
drwxr-xr-x   2 root     sys          1024 Dec 22  1996 rc2.d
drwxr-xr-x   2 root     sys          1024 Dec 22  1996 rc3.d
drwxr-xr-x   2 root     sys          1024 Dec 22  1996 rc4.d
drwxr-xr-x   2 root     sys          1024 Dec 22  1996 rc5.d
drwxr-xr-x   2 root     sys          1024 Dec 22  1996 rc6.d
drwxr-xr-x   2 root     sys          1024 Dec 22  1996 rcS.d
```

如果是LINUX系统，那么……

```
$ pwd
```

China-pub.com
下载

（续）

```
/etc/rc.d
$ ls
init.d      rc.local    rc0.d       rc2.d       rc4.d       rc6.d
rc          rc.sysinit  rc1.d       rc3.d       rc5.d
```

如果我们使用cd命令进入这些rcN.d目录，会发现这些目录中的rc.scripts实际上是一些链接。

```
$ pwd
/etc/rc.d/rc2.d
$ ls -l
lrwxrwxrwx   1 root root 16 Dec 3 15:16 K87ypbind -> ../init.d/yd
lrwxrwxrwx   1 root root 17 Dec 3 15:10 K89portmap -> ../init.d/p
lrwxrwxrwx   1 root root 17 Dec 3 15:07 S01kerneld -> ../init.d/d
...
...
```

28.2 确定当前的运行级别

本章不是针对系统管理员的，但是作为shell编程者，应当了解rc.scripts是什么，它们是被怎样放置到运行级别配置目录中的。顺便说一下，如果想知道当前的运行级别，可以用下面的命令：

```
 $ who -r
    .        run-level 4  Apr 22 13:26    4    0    3
```

在‘run level’后面的数字就是当前的运行级别。后面的时间是系统最近一次重启动的时间。

如果是LINUX系统，那么……

```
$ runlevel
2 3
```

第一列表示系统的前一个运行级别，第二列表示系统当前的运行级别，在这里是 3。

28.3 快速熟悉inittab

运行级别目录中含有一系列启动服务的脚本。这里的“服务”可以是守护进程、应用程序、服务器、子系统或脚本进程。在系统启动的过程中，将会启动一个名为 init的进程（它是系统中所有进程的祖先）。它所要完成的一部分工作就是看看需要启动哪些服务，应当缺省地进入哪一个运行级别。它通过查看一个名为 inittab的配置文件来获得上述信息，该配置文件位于/etc目录下。init进程还按照该文件中的设置加载特定的进程。如果需要编辑这个配置文件，一定要先做一个备份。如果该文件被破坏或出现“降级”错误，系统将无法正常启动，到那时，将不得不进入单用户模式并修正该文件。

inittab文件所包含的域具有严格的格式。该文件中每个条目的格式为：

```
id:rstart:action:process
```

其中，id域是相应进程的唯一标识。

rstart域所包含的数字表示运行该进程的级别。

action域告诉init进程如何对待process所对应的进程。这里可以有很多种动作，但是最常

China-pub.com 下载

见的是wait和respawn。wait意味着当进程启动后等待它结束。 respawn则意味着如果该进程不存在，则启动相应的进程，如果它存在，那么只要它一掉下来就立即重新启动它。

process域包含了实际要运行的命令。下面是 inittab文件的一部分。

```
$ pg /etc/inittab
id:3:initdefault:

# System initialization.
si::sysinit:/etc/rc.d/rc.sysinit
# run level 0
l0:0:wait:/etc/rc.d/rc 0
# run level 1
l1:1:wait:/etc/rc.d/rc 1
# run level 2
l2:2:wait:/etc/rc.d/rc 2
# run level 3
l3:3:wait:/etc/rc.d/rc 3
# run level 4
l4:4:wait:/etc/rc.d/rc 4
# run level 5
l5:5:wait:/etc/rc.d/rc 5
# runlevel 6
l6:6:wait:/etc/rc.d/rc 6
#Run gettys in standard runlevels
1:12345:respawn:/sbin/mingetty tty1
2:2345:respawn:/sbin/mingetty tty2
3:2345:respawn:/sbin/mingetty tty3
4:2345:respawn:/sbin/mingetty tty4
5:2345:respawn:/sbin/mingetty tty5
6:2345:respawn:/sbin/mingetty ttyS1 vt100
```

该文件的第一行是系统缺省的运行级别，这里是级别 3，一般都是这样。

以数字10到16开始的行启动或停止该运行级别所对应的全部运行级别脚本。例如，该文件中有这样一行：

```
l5:5:wait:/etc/rc.d/rc 5
```

它的意思是，在运行级别 5应该以参数5执行脚本/etc/rc.d/rc，即/etc/rc.d/rc执行/etc/rc.d/rc5.d目录中的所有脚本。

在上述文件的最后一行，在运行级别 2、3、4和5，该进程将会始终存在，即使暂时掉下来，大概也不会超过 1s。这一始终存在的进程是串口 ttyS1上的mingetty。该命令含有一个参数，即终端类型为vt100。

28.4 运行级别

init进程在系统完全就绪之前所做的最后几项工作之一就是执行缺省运行级别所包含的所有脚本。该进程是通过 /etc/rc.d/rc或/etc/rc.init来启动这些脚本的。它的作用是首先杀死该运行级别所包含的进程再启动这些进程。

但是它怎么知道该启动或停止哪些服务呢？ rc或rc.init脚本将会使用for循环来依次查看相应运行级别目录中的文件，给每一个链接名以 K开头的相应脚本赋予参数stop；给每一个链接名以S开头的相应脚本赋予参数 start。在运行级别切换时，上述脚本也会完成同样的工作，只不过根据相应的运行级别来启动或停止对应的脚本。

rcN.d目录中的脚本只是一些链接——真正的脚本保存在其他的目录中。它们通常都放置

China-pub.com
下载

在/usr/sbin/init.d或/etc/init.d目录中。

> 如果是LINUX系统，那么……
>
> /etc/rc.d/init.d

在这个目录中含有一些能够启动或停止某一服务的脚本。这些脚本的名字最好能够表示出它所实现的功能，形如rc.<功能>，其中rc表示运行命令（run command）或运行控制（run control），或者就像某些系统管理员所称的那样“真正关键的”（‘real crucial’）。

下面是这类文件的部分列表。

```
$ ls
rc.localfs        rc.halt       rc.reboot   rc.syslogd   rc.daemon
...
```

一般来说，rc.scripts都应当能够接受这样的参数：

rc.name stop：停止该服务。

rc.name start：启动该服务。

可选的参数包括restart和status。其他任何参数都应当给出相应的用法说明。注意，可以手工运行这些脚本。

现在我们已经知道运行级别脚本应当具有什么样的功能，下一步就是把它们放置在相应的rcN.d目录中。不过在此之前我们先来了解一下系统运行级别。

28.4.1　各种运行级别

系统含有七种运行级别(见表28-1)。不同的系统在某些运行级别上稍有差别。

在将一个脚本放置在不同的运行级别目录中之前，首先应当弄清楚打算在哪一个运行级别启动或停止相应的服务？一旦弄清楚这一点，就可以接着进行下面的步骤了。

表28-1　各个运行级别的用途

运行级别0	启动和停止整个系统
运行级别1	单用户或管理模式
运行级别2	多用户模式；部分网络服务被启动。有些系统将其作为正常运行模式，而不是级别 3
运行级别3	正常操作运行模式，启动所有的网络服务
运行级别4	用户定义的模式，可以使用该级别来定制所需要运行的服务
运行级别5	有些UNIX操作系统变体将其作为缺省X-windows模式，还有些系统把它作为系统维护模式
运行级别6	重启动

28.4.2　运行级别脚本的格式

rcN.d目录中的脚本都是一些链接，这样是为了省去不必要的副本。这些链接的格式为：

```
Snnn.script_name
```

或

```
Knnn.script_name
```

其中，

S：代表启动相应的进程

K：代表杀死相应的进程

nn：是00至99的两位数字，不过在有些系统中是000至999三位数字。在不同目录中的链

China-pub.com 下载

接应采用同一数字。例如，如果某个服务在 rc3.d中启动时名为S45.myscript，那么如果希望它在rc2.d中启动，应当使用链接名 S45.myscript。

script_name：相应脚本的文件名，根据所在操作系统的不同，它们可能位于下列目录中：

```
/usr/sbin/init.d
/etc/rc.d
/etc/init.d
```

当init进程调用相应的运行级别脚本时，杀进程按照从高到低的 K序号进行，即K23,myscript K12.named；而启动进程按照从低到高的序号进行。如果使用的是 LINUX系统，K序号将按照从高到低的顺序执行。

28.4.3 安装运行级别脚本

如果想要安装自己的运行级别脚本，必须：

- 编写该脚本，确保它符合调用标准。
- 确信它能够启动或终止相应的服务。
- 将该脚本放置于(取决于操作系统)/etc/init.d或/usr/sbin/init.d或/etc/rc.d中。
- 在相应的rcN.d目录中按照合理的命名方式创建链接。

下面的脚本能够启动或停止一个名为 rc.audit的审核应用程序。该服务运行于级别3、5、4，停止于级别 6、2、1。通过查看 rcN.d中的条目，我们发现序号 35空闲，于是就使用该序号。实际上，系统并不对使用已占用的序号作任何检查。

下面就是这个脚本。可以看到，该脚本使用了一个简单的 case语句来接收start和stop参数。

```
$ pg rc.audit
#!/bin/sh
# rc.audit start| stop
# script to start or stop zeega's audit application
#
case "$1" in
start)
  echo -n "Starting the audit system...."
  /apps/audit/audlcp -a -p 12
  echo
  touch /var/lock/subsys/rc.audit
  ;;
stop)
  echo -n "Stopping the audit system...."
  /apps/audit/auddown -k0
  echo
  rm -f /var/lock/subsys/rc.audit
  ;;
restart)
  $0 stop
  $0 start
  ;;
*)
  echo "To call properly..Usage: $0 {start|stop|restart}"
  exit 1
  ;;
esac
```

```
exit 0
```

如果是LINUX系统，那么……

有些LINUX变体在启动服务时要求创建一个锁文件。如果没有锁文件，在杀死该脚本时就可能会出现问题。

start选项将使该审核进程启动相应的审核系统，而 stop选项将使它终止该系统的运行。当然，在将自己的运行级别脚本放置在 init.d目录中之前，应该首先对该脚本进行测试。

```
$ rc.audit
To call properly..Usage:./rc.audit {start|stop|restart}
$ rc.audit start
Starting the audit system....
```

让我们假定该脚本已经通过了测试。它能够正确地启动和停止审核服务。现在我们把该脚本放置在相应的运行级别目录中。

在本系统中，rcN.d目录位于/etc/rc.d目录下，而我的运行级别脚本保存在 /etc/rc.d/init.d目录下。如果系统目录结构与上面的不同，那么需要对下面的命令作相应的调整。

我们首先启动该脚本——记住启动脚本所使用的链接名是以 S打头的。

```
$ pwd
/etc/rc.d/rc3.d
$ ln -s../init.d/rc.audit S35rc.audit

$ ls -l
...
lrwxrwxrwx   1 root   root 27 May  8 14:37 S35rc.audit -> ../init.d/
rc.audit
...
```

我们已经创建了相应的链接。ls -l命令的结果显示该链接指向 /etc/init.d/rc.audit文件。我本应该在链接命令中给出全路径，不过没有这个必要。现在我只要进入其他的相关目录 (rc4.d和rc5.d)使用同样的命令就可以启动其他相应的服务。

如果希望停止某个脚本的运行，可以使用如下命令：

```
$ pwd
/etc/rc.d/rc6.d
$ ln -s../init.d/rc.audit K35rc.audit
$ ls -l
...
lrwxrwxrwx   1 root   root  27 May  8 14:43 K35rc.audit -> ../init.d/
rc.audit
...
```

在其他相关目录中，也可以如法炮制，停止相应的审核服务。现在当系统重启动时 (运行级别6)，它将被停止；在运行级别切换到 2或1时也是如此。该服务在运行级别 4或5中同样也会被启动。

28.5 使用inittab来启动应用程序

我们还可以用其他的方法来启动应用程序。可以通过在 inittab文件中加入相应的条目来做到这一点。在我所管理的有些系统上，我就使用了这种方法，这倒不是因为这些系统中没有运行级别目录，而是由于我有几个用于系统检查的脚本需要在系统刚刚就绪之后运行。使用inittab是实现上述功能的理想途径。

China-pub.com 下载

这里我们给出一个例子，我打算在系统运行在级别 3时运行我的一个磁盘镜像检查脚本。首先我确定该脚本能够正确运行，然后对 inittab文件做备份。

```
$ cp /etc/initab /etc/inittab.bak
```

接下来编辑inittab文件，在该文件末尾加入这样一个条目：

```
# disk checker script, let's see if any of the mirrors are broken.
rc.diskchecker:3:once:/usr/local/etc/rc.diskchecker > /dev/console 2>&1
```

保存并退出。

上面的一条意思是：

行首的rc.diskchecker是该进程在运行级别3中的唯一标识。该进程只运行一次。所要运行的脚本是/usr/local/etc/rc.diskchecker，所有的输出都被送到控制台。

28.6 启动和停止服务的其他方法

如果不想把/etc/inittab文件弄得过于杂乱，还有其他的方法可以实现启动和停止服务的功能。大多数系统都含有一个名为rc.local的文件，一般来说也是位于/etc目录下。该脚本文件将在inittab和运行级别脚本之后运行。可以在该文件中加入任何命令，或从中调用最习惯用的启动脚本。

有些系统还在/bin目录下(更多的是在/usr/sbin目录下)含有一个名为shutdown的脚本文件。可以使用它来关闭某些服务。

28.7 小结

运行级别的确是一个系统管理问题。本章的目的在于：使你了解在系统启动时，如何按照需求灵活地控制各种服务和脚本的启动。

这还意味着在系统重启动时，能够手工启动和停止某些服务。

China-pub.com
下载

第29章 cgi 脚 本

现在差不多每个人的PC上都安装了Web服务器，在这样一本关于shell编程的书中似乎很有必要包含一章关于cgi脚本的内容。

本章包含以下内容：

- 基本cgi脚本。
- 使用服务器端内嵌(Server Side Includes,SSI)。
- get方法。
- post方法。
- 创建交互式脚本。
- 能够自动重载Web页面的cgi脚本。

运行Web服务器并不一定需要有网络环境，可以在本地主机上运行它。这里，我们假定你已经安装了Web服务器(apache、Cern等等)以及浏览器(Netscape、Internet Explorer等等)。另外，该服务器应当允许运行cgi脚本。一般来说缺省值是禁止运行cgi脚本的，要运行，只要将配置文件中相应的一行注释掉即可。后面我们会更详细地讨论这一问题。

如何安装并配置Web服务器已经超出了本书的讨论范围，不过我认为只需 20分钟就可以安装并运行一个Web服务器。本章中的例子运行于apache Web服务器下，我所使用的浏览器为Netscape。

本章不打算深入探讨有关HTML或Web的细节问题，因为市面上已经有大量关于这方面的书籍。另外，如果要深入探讨HTML的话，还要花费数章的笔墨。

29.1 什么是Web页面？

Web页面或文挡是包含有HTML标记的文件。当浏览器连接到一个Web页面上时，浏览器就会根据相应的HTML标记来显示该页面。Web页面中可以含有非常丰富的信息，它可以包含指向其他页面的链接、各种色彩、高亮标题、各种字体、直线、表格，还可以包含图像和声音。

Web页面可以分为两类：动态的页面和静态的页面。静态的页面是用于显示信息或下载文件。而动态的页面是交互型的，它们可以按照你所提供的信息产生相应的结果。动态页面还可以用于显示实时变化的信息，如股票价格，或用于完成某些监视任务。如果想要执行这种类型的处理，就需要编写脚本。

如果一个Web服务器能够交换信息脚本，那么它必须支持一种被称为公共网关接口的协议，即大家所熟悉的cgi。

29.2 cgi

cgi是一种规范，它规定了获取信息的脚本如何从服务器中取得信息或向服务器中写入信息。这种脚本或cgi脚本可以用任何语言来实现。最为流行的是 Perl语言，不过你将会发现，

China-pub.com 下载

也可以用普通的shell脚本来实现（见图29-1）。

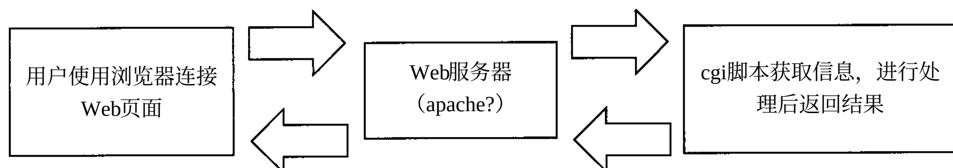


图29-1 浏览器和服务器可以通过cgi来交换信息

29.3 连接Web服务器

可以使用统一资源定位符(URL)连接Web服务器。URL包含两部分信息：

- 协议。
- 地址和数据。

其中，协议包括http、ftp、mailto、file、telnet和news。这里我们只关心http协议(超文本传输协议)。

地址一般是DNS域名或服务器主机名，也可以是 IP地址。其他数据可以是你所要访问文件的实际路径名。

所有的连接都基于TCP协议之上，缺省的端口号为80。

如果Web服务器在你的本地主机上，而相应的主页为 index.html，那么可以使用下面的URL：

```
http://localhost/index.html
```

一般来说，index.html是缺省下载的文件，即该页面是你的 Web服务器的缺省页。这样，你可以只输入如下的URL：

```
http://localhost/
```

29.4 cgi和HTM脚本

当浏览器发出下载页面的请求时， Web服务器将会对收到的URL进行分析。如果其中含有cgi-bin，服务器将打开一个连接，通常是连接相应 cgi脚本的管道。该cgi脚本所有的输入输出都将通过该管道。如果该 cgi脚本用于显示Web页面，那么它的输出中必须要包含必要的HTML标记，这样该页面才能够按照服务器所能够理解的格式被显示出来，因此我们有必要了解一些HTML的知识。Web服务器将该页面返回给发出请求的浏览器显示出来。表 29-1列出了一些常用的HTML标记。

29.4.1 基本cgi脚本

所有的cgi脚本都应当位于Web服务器的cgi-bin目录中，不过在不同的服务器中该目录会有所不同。可以通过查看配置文件 srm.conf中ScriptAlias一段来改变该目录的位置，并允许该服务器运行 cgi脚本。所有的脚本文件名都应以 .cgi做后缀。而其他 Web页面都位于 html或htdocs目录下，并且带有.html后缀。所有的脚本都应具有这样的权限。

```
chmod 755 script.cgi
```

所有Web页面连接的缺省用户身份为 nobody，不过可以通过配置 httpd.conf文件来改变这

China-pub.com
下载

一设置。尽管我曾经说过本章并不是关于 Web服务器配置的，不过正好可以借这个机会检查一下passwd文件，看看nobody用户的登录是否被禁止。如果想防止任何用户以 nobody的身份从某一个物理端口上登录，只要在 /etc/passwd文件中该用户口令域的第一个字符前面加入一个星号*即可。

表29-1 基本HTML标记

<HTML></HTML>	HTML文挡的开头和结束
<HEAD></HEAD>	标题部分的开头和结束
<TITLE></TITLE>	主题的开头和结束
<BODY></BODY>	页面部分的开头和结束
<Hn></Hn>	不同层次的标题，1代表最大的字体
<P></P>	段落的开头和结束
 	换行
<HR>	水平线
<PRE></PRE>	预定义格式文本的开头和结束，其中的所有跳格、行都保持原样。
<B></B>	黑体
<I></I>	斜体
<OL></OL>	有序号的列表
<A HREF=url>link</A>	超文本链接
<FORM></FORM>	表单
METHOD	post或get方法
ACTION	地址
<INPUT...>	数据输入
NAME	变量名
SIZE	以字符计的文本框的宽度
TYPE	复选框、单选框、复位、提交
<SELECT...>	下拉式菜单
NAME	变量名
SIZE	要显示的列表的项数
<OPTION VALUE>	将用户选择的值返回给NAME变量
</SELECT>	结束列表框

如果脚本不能正常工作，应当首先查看错误日志，因为所有的错误都记录在这些日志中。如果使用apache作为Web服务器，那么相应的日志文件位于/etc/httpd/logs或/usr/local/apache/logs目录下，这取决于Web服务器的安装路径。cgi脚本可以在命令行方式下运行测试，当然这时只能看到文本形式的输出，但是这种输出结果有助于调试该脚本。

现在我们就来创建一个cgi脚本。把下面的内容输入到一个名为 firstpage.cgi的文件中，并保存在cgi-bin目录下。不要忘记该文件应当具有权限 755。

```
$ pg firstpage.cgi
#!/bin/sh
# firstpage.cgi
# display a page with text
echo "Content-type: text/html"
echo ""
echo "<HTML>"
echo "<H1><CENTER> THIS IS MY FIRST CGI PAGE</CENTER></H1>"
echo "<HR>"
echo "<H2><CENTER>STAND-BY TO STAND-TO!</CENTER></H2>"
echo "</HTML>"
```

China-pub.com 下载

如你所知，第一行表示shell解释器的路径。第一个echo命令行告诉服务器这是一个MIME题头；第二行echo命令行用于显示一个空行。如果在MIME题头后面没有一个空行，cgi脚本的输出将无法正确显示。

接着，echo命令输出了一个<HTML>标记，它告诉浏览器整篇文挡应以HTML格式显示。HTML文挡可以显示从最大的<H1>到<Hn>的若干种不同字体。为了不过于费眼，<H6>是通常可接受的最小字体。这里我们把文字居中，这样看起来更舒服一些。接下来，我们显示一条水平线。最后，我们用<H2>字体和<CENTER>标记居中显示了这样一行："Stand-By To Stand-To"，该脚本的输出以</HTML>标记结束。

如果忘记了某一个结束标记，不要紧——你会很快发现这一点，因为在你试图浏览该页面时，一些标记将会在浏览器中显示出来。

现在如果想浏览该页面，可以在浏览器的URL框中输入这样一行：

```
http://your_server/cgi-bin/firstpage.cgi
```

输入时用实际的服务器名来替代上面一行中的your_server。

如果你的机器已经联网，而你得到"DNS查找失败"的错误提示，那可能是由于浏览器在Internet上查找你刚刚编写的页面。不妨查看一下浏览器的连接选项；它可能设置为忽略本机代理，在此处键入主机名并重新运行浏览器即可。

图29-2显示了我们刚刚编写的页面。

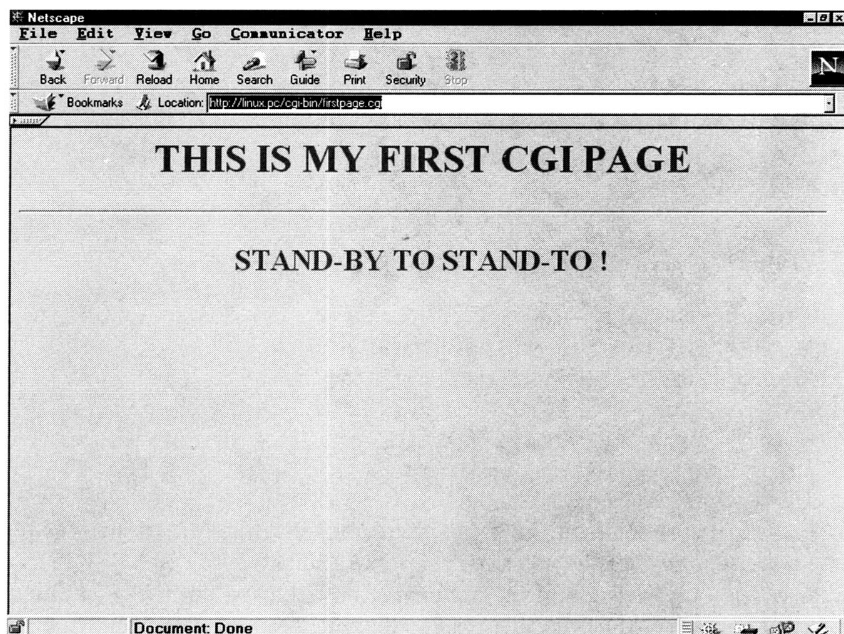


图29-2 脚本firstpage的输出

29.4.2 显示shell命令输出

现在我们在脚本中加上一条shell命令，这样就可以在浏览器中显示该命令的输出。

我们将显示当前登录的用户数，这通过将who命令的输出经管道传递给wc命令就可以实现。还将显示当前的日期。

China-pub.com 下载


```
$ pg pagetwo.cgi
#!/bin/sh
# pagetwo.cgi
# display a page using the output from a unix command
MYDATE=`date +%A" "%d" "%B" "%Y`
USERS=`who |wc -l`
echo "Content-type: text/html"
echo ""
echo "<HTML>"
echo "<H1><CENTER> THIS IS MY SECOND CGI PAGE</CENTER></H1>"
echo "<HR>"
echo "<H2><CENTER>$MYDATE</CENTER></H2>"
echo " Total amount of users on to-day is :$USERS"
echo "<PRE>"
if [ "$USERS" -lt 10 ]; then
  echo  " It must be early or it is dinner time"
  echo  " because there ain't many users logged on"
fi
echo "</PRE>"
echo "</HTML>"
```

在这个脚本的开头，我们取得了当前的日期和连接数。日期居中显示。变量 USERS也被显示出来。通过一个if语句来判断用户数是否少于10，如果是，则显示这样的信息“现在还早或现在是晚餐时间”。

标记<PRE>保留其间的所有空格和跳格。如果希望显示系统命令的输出，如用 df命令显示文件系统的情况，或只是使用简单的echo命令，那么应当使用<PRE>标记。在本例中我本来没有必要使用<PRE>标记，但我想我还是应该及早介绍它的这种应用，这样即使读者今后遇到这样的问题，也不会感到迷惑。要显示该页面，可以使用如下的URL：

```
http://your_server/cgi-bin/pagetwo.cgi
```

输入时用实际的服务器名来替代上面一行中的your_server。

图29-3显示了刚才编辑的页面。

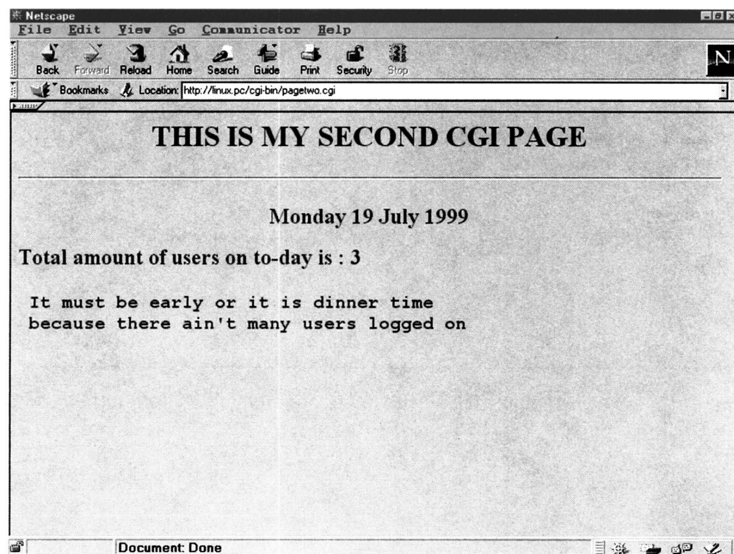


图29-3 脚本pagetwo.cgi的输出

29.4.3 使用SSI

使用cgi脚本产生一个Web页面并显示少量信息有点杀鸡用牛刀。例如我们用 cgi脚本产生了一个页面而仅仅是为了显示当前日期。如果我们能够把 cgi脚本嵌入到HTML文挡中，让该脚本的输出显示在普通的页面中岂不更好？这样做是可行的，下面我们就将讨论这一内容。

为了内嵌cgi脚本，我们必须使能服务器端内嵌(SSI)。这样，在显示一个页面时，它将会把SSI命令替换为相应命令或脚本的输出。一些含有服务器和命令信息的特殊环境变量也是全局可见的。

为了让服务器能够知道替换文挡中的相应 SSI命令，需要使能SSI，应当去掉配置文件中含有server-passed的一行开头的注释符，在apache上是这样的：

```
Addhandler server-passed.shtml
Addtype   text/htm   shtml
```

需要重新启动Web服务器软件，使用kill -1命令使该服务器重新读入配置文件。含有 SSI的页面应当具有后缀.shtml，而不是.html。

29.4.4 访问计数器

现在让我们来创建一个能够显示访问次数的页面。你肯定见过类似的页面："你是第n个访问本站点的用户"。我们还将显示该页面的最新更改时间。

不要忘记将下面的脚本放置在cgi-bin目录中，起名为hitcount.cgi。

```
$ pg hitcount.cgi
#!/bin/sh
# hitcount.cgi
# hit page counter for html <cgi>
# counter file must be chmod 666
counter=../cgi-bin/counter
echo "Content-Type: text/html"
echo ""
read access < $counter
access=`expr $access + 1`
echo $access
echo $access >$counter
```

如你所见的，该脚本读入../cdi-bin/counter文件中的值并将其赋给变量access，将该变量加1，显示该变量，最后将新的值写回到../cgi-bin/counter文件中。

现在我们创建一个名为counter的文件。我们只需在其中放置一个初始值，这里我们取 1。于是，操作步骤为：创建一个名为counter的文件并写入1(不要加引号)，然后保存并退出。

由于每个用户都需要使用该文件，文件属主、同组用户及其他用户都应对其具有读和写的权限。

```
$ chmod 666 counter
```

现在还需要在Web根目录下(该目录通常是放置所有其他HTML文挡的地方，一般是htdocs或html目录)创建一个相应的.shtml文件。下面就是该文件，千万不要忘记带.shtml：

```
$ pg main.shtml
<!- main.shtml>
<!- this is a comment line>
```

```
<HTML>
<H4> Last modified: <!--#echo var="LAST_MODIFIED" -->
</H4>
<HR>
<H1><CENTER> THE MAY DAY OPERATIONS CENTER </H1>
<H2>Stand-by to Stand-to
<HR>
This page has been visited <!--#exec cgi="/cgi-bin/hitcount.cgi"--> times
</CENTER>
</H2>
<HR>
</HTML>
```

在使用SSI时LAST_MODIFIED变量及其他变量是全局可见的。可以通过访问 apache的Web站点(www.apache.org)得到所有在使用SSI时全局可见的特殊变量的详细描述。

我们来看下列的SSI命令：

```
This page has been visited <!--#exec cgi="/cgi-bin/hitcount.cgi"--> times
```

它的一般形式为：

```
<!--#command argument="value"-->
```

在本例中， cgi脚本hitcount是这样运行的：

命令是exec。

参数为cgi。

其中的value是要运行的脚本。

我已经改变了配置文件，把这个页面作为主页，而不是 index.html。不过你仍然可以通过在URL中使用全路径来访问index.html页面。

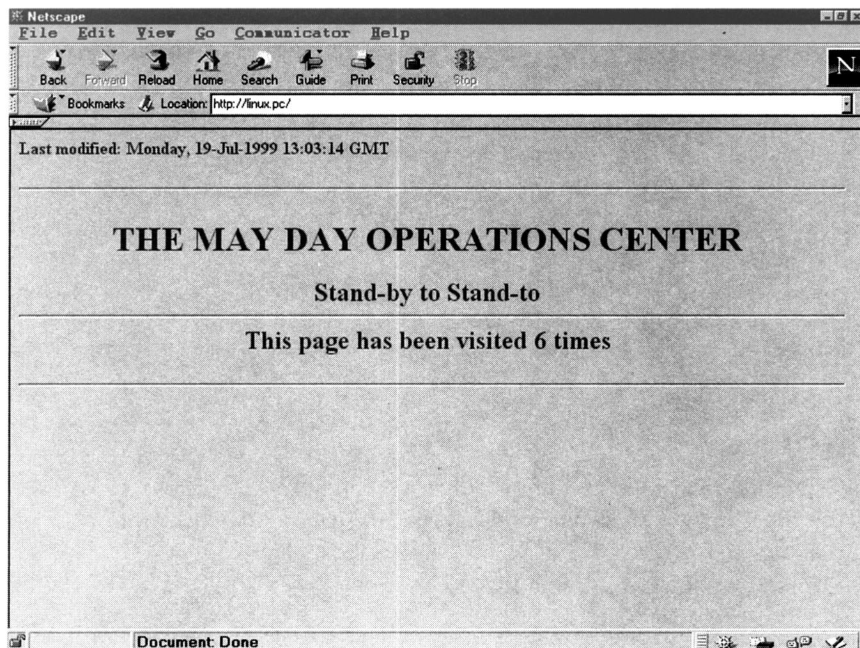


图29-4 含有点击计数器的页面

China-pub.com 下载

如果想改变缺省页，可以编辑srm.conf文件。该文件中含有这样一行：

```
DirectoryIndex
```

你会看到文件名index.html显示在该条目中。把它改成所希望的缺省页即可。不要忘记关闭并重新启动Web服务器以使该改动生效。

要访问该脚本，可以在浏览器的URL框中输入：

```
http://<server_name>/main.shtml
```

或

```
http://<server_name>
```

(如果该页面是缺省页的话)。

图29-4显示了刚才创建的页面；可以刷新该页面观察计数器的增加。注意， LAST_MODIFIED变量的值也被显示出来。

可以通过在cron中加入一行，每天夜里向counter文件中写入1来复位该计数器的值。

29.4.5 使用一个链接来显示当前Web环境变量

在执行cgi脚本时，有若干环境变量可用。可以通过 env或set命令看到绝大多数的变量。我们在main.shtml文挡中创建一个指向相应cgi脚本的链接，该脚本能够显示这些变量。

下面是我们使用的链接：

```
<A HREF="/cgi-bin/printenv.cgi">Environment</A>
```

其中A HREF是链接标记的头。相应的地址包含在双引号中。单词 Environment是要显示出来的单词，用户将点击这里。 </A>是链接标记的尾。

下面就是main.shtml文挡：

```
$ pg main.shtml

<HTML>
<! this is a comment line>
<! main.shtml>
<H4> Last modified: <!--#echo var="LAST_MODIFIED" -->
</H4>
<HR>
<CENTER>
<H1> THE MAY DAY OPERATIONS CENTER </H1>
<H2> Stand-by to Stand-to
<HR>
This page has been visited <!--#exec cgi="/cgi-bin/hitcount.cgi"--> times
<HR>
To see your environment settings just click
     <A HREF="/cgi-bin/printenv.cgi" >here</A>
</CENTER>
</H2>
<HR>
</HTML>
```

下面是被调用的脚本 printenv.cgi，它使用env命令显示环境变量。在这里，我们使用<PRE>标记来保持该命令输出中的空格和跳格不变。

```
$ pg printenv.cgi
```

```
#!/bin/sh
# printenv.cgi
# print out the Web server env using the command env settings,
echo "Content-type: text/html"
echo ""
echo "<HTML><PRE>"
env
echo "</PRE></HTML>"
```

图29-5显示了该页面。

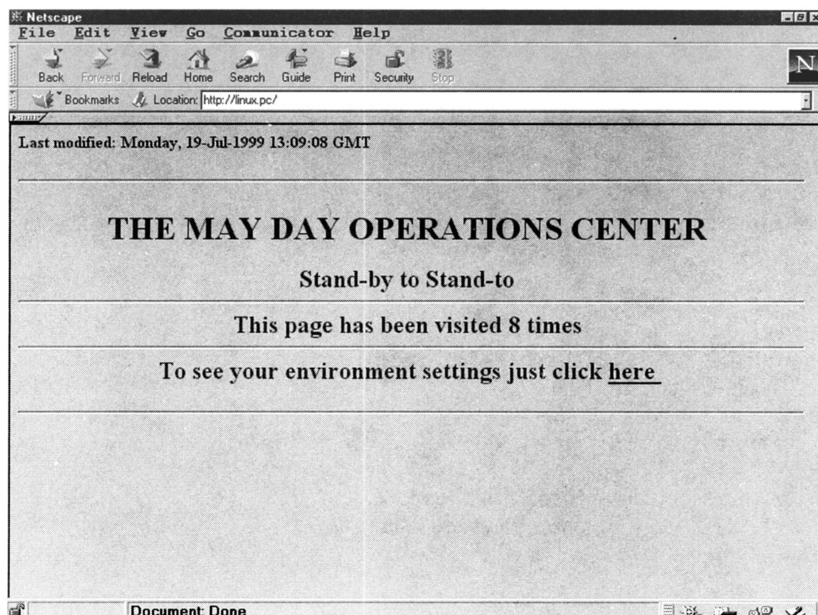


图29-5 含有显示环境变量链接的页面

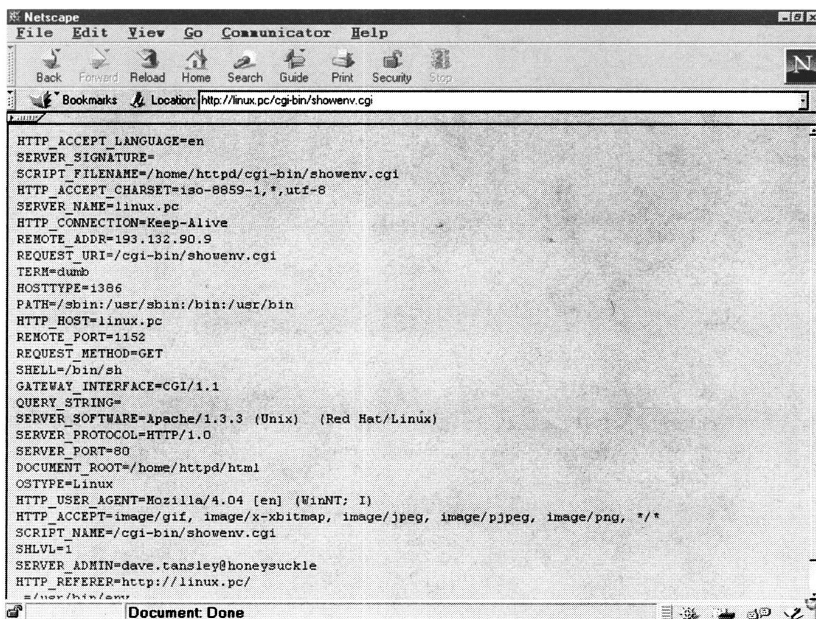


图29-6 显示当前环境变量的页面

China-pub.com 下载

当你点击该链接时，将会显示出当前的环境设置 (见图29-6)。你在自己机器上所看到的可能与此有所不同。在运行不同的脚本时，一些相应的环境变量值将会随之改变。

29.4.6 其他常用的环境变量

表29-2列出了最常用的cgi环境变量。其中有些变量可以用 env或set命令显示出来。

表29-2 常用的cgi Web服务器变量

DOCUMENT ROOT	Web服务器的主目录，是放置HTML文挡的地方
GATEWAY_INTERFACE	cgi的版本
HTTP_ACCEPT	可接受的各种MIME类型
HTTP_CONNECTION	缺省的HTTP连接
HTTP_HOST	本地主机名
HTTP_USER_AGENT	客户端浏览器
REMOTE_HOST	远程主机
REMOTE_ADDR*	远程主机的IP地址
REQUEST_METHOD	传递信息的方法
SCRIPT_FILENAME	cgi脚本的绝对路径
SCRIPT_NAME	cgi脚本的相对路径
SERVER_ADMIN	Web服务器管理员的邮件地址
SERVER_NAME	服务器的主机名、DNS或IP地址
SERVER_PROTOCOL	连接所使用的协议
SERVER_SOFTWARE	Web服务器软件名
QUERY_STRING	get方法所传递的数据
CONTENT_TYPE	MIME类型
CONTENT_LENGTH	post方法所传递的字节数

*严格来讲，这是你连接至Internet的网关地址。

为了显示这些变量，可以把它们放在一个 cgi脚本中，这样，在需要使用某个变量值时随时都可以调用该脚本。

```
$ pg evncgi.cgi
#!/bin/sh
# envcgi.cgi
# print out the web server env
echo "Content-type: text/html"
echo ""
echo "<HTML><PRE>"
echo "CGI Test ENVIRONMENTS"
echo "SERVER_SOFTWARE = $SERVER_SOFTWARE"
echo "SERVER_NAME = $SERVER_NAME"
echo "GATEWAY_INTERFACE = $GATEWAY_INTERFACE"
echo "SERVER_PROTOCOL = $SERVER_PROTOCOL"
echo "SERVER_PORT = $SERVER_PORT"
echo "REQUEST_METHOD = $REQUEST_METHOD"
echo "HTTP_ACCEPT = $HTTP_ACCEPT"
echo "PATH_INFO = $PATH_INFO"
echo "PATH_TRANSLATED = $PATH_TRANSLATED"
echo "QUERY_STRING = $QUERY_STRING"
echo "SCRIPT_NAME = $SCRIPT_NAME"
echo "REMOTE_HOST = $REMOTE_HOST"
echo "REMOTE_ADDR = $REMOTE_ADDR"
echo "REMOTE_USER = $REMOTE_USER"
echo "AUTH_TYPE = $AUTH_TYPE"
```

```
echo "CONTENT_TYPE = $CONTENT_TYPE"
echo "CONTENT_LENGTH = $CONTENT_LENGTH"
echo "</PRE></HTML>"
```

29.5 get和post方法简介

到现在为止，我们只是向屏幕上输出。要想从用户那里得到信息，我们需要使用表单，这也是cgi为什么如此流行的原因。你需要有获得用户输入的能力。有了表单就可以显示文本框、下拉式列表框和单选框。

在用户通过键入或选择向表单输入了一些信息之后，他可以点击发送按钮将这些信息发送给某个脚本，在这里是cgi脚本。我们需要使用get或post方法来收集这样的信息。

29.5.1 get方法

任何表单的缺省操作都是get方法。get方法是从静态HTML页面获取文件的方法。

当用户点击“提交”按钮时，用户选择的信息将以编码字符串的形式附加在服务器 URL的后面。服务器环境变量QUERY_STRING保存了编码字符串。变量REQUEST_METHOD保存了该表单所使用的方法。

1. 创建一个简单的表单

现在让我们来创建一个简单的表单，在main.shtml文档中创建一个指向booka.cgi脚本的链接。

在main.shtml文档中上一次创建的链接后面增加这样一行：

```
<BR>
Basic form using GET method <A HREF="/cgi-bin/booka.cgi" >Form1</A>
```

现在就按照下面的内容创建booka.cgi文件，不要忘记把它放在cgi-bin目录中。

```
$ pg booka.cgi
#!/bin/sh
# booka.cgi
echo "Content-type: text/html"
echo ""
echo "<HTML>"
echo "<BODY>"
# call booka_result.cgi, when user hits sent!
echo "<FORM action="/cgi-bin/booka_result.cgi" METHOD=GET>"

echo "<H4> CGI FORM</H4>"
# text box, input assigned to variable name 'contact'
echo "Your Name: <INPUT NAME=contact SIZE=30><BR><BR>"
# drop down menu selection assigned to variable name 'film'
echo "<SELECT NAME=film>"
echo "<OPTION>--  Pick a Film  --"
echo "<OPTION>A Few Good Men"
echo "<OPTION>Die Hard"
echo "<OPTION>Red October"
echo "<OPTION>The Sound Of Music"
echo "<OPTION>Boys In Company C"
echo "<OPTION>Star Wars"
echo "<OPTION>Star Trek"
```

China-pub.com 下载

```
echo "</SELECT>"
# drop down menu selection assigned to variable name 'actor'
echo "<SELECT NAME=actor>"
echo  "<OPTION>-- Pick Your Favourite Actor --"
echo "<OPTION>Bruce Willis"
echo "<OPTION>Basil Rathbone"
echo "<OPTION>Demi Moore"
echo "<OPTION>Lauren Bacall"
echo "<OPTION>Sean Connery"
echo "</SELECT>"
echo "<BR><BR>"
# check box variable names are 'view_cine' and 'view_vid'
echo "Do you watch films at the..<BR>"
echo "<INPUT TYPE="Checkbox" NAME=view_cine> Cinema"
echo "<INPUT TYPE="Checkbox" NAME=view_vid> On Video"
echo "<BR><BR>"
# input assigned to variable name 'textarea'
echo "Tell what is your best film, or just enter some comments<BR>"
echo "<TEXTAREA COLS="30" ROWS="4" NAME="textarea"></TEXTAREA>"

echo "<BR><INPUT TYPE=Submit VALUE="Send it">"
echo "<INPUT TYPE="reset" VALUE="Clear Form">"

echo "</FORM>"
echo "</BODY>"
echo "</HTML>"
```

我们在上述表单的操作项中设定：一旦用户按下“发送”按钮，脚本 booka_resullt.cgi将被执行。我们现在讨论get方法。

上面的表单将会显示两个文本框、两个下拉式列表框和一个复选框。

输入姓名的文本框宽度为30个字符，相应的输入被赋给变量 contact。

第一个下拉式列表框让用户选择最喜爱的电影，选择项被赋给变量 film。

第二个下拉式列表框让用户选择最喜爱的演员，选择项被赋给变量 actor。

用户可以通过点击选择某一个复选框或两个都选。相应的逻辑值分别保存在变量view_cine和变量view_vid中。如果用户选择某一个复选框，那么相应的变量的值为 on。

区域型文本框允许用户输入超过一行的文本，而不是像标准文本框那样只能输入一行 (在本例中是4行，每行宽30个字符)，所有的输入都将赋给变量textarea。

发送数据时需要使用submit作为输入类型。用户可以点击clear按钮清除当前的表单。

创建如下的cgi脚本，命名为booka_result.cgi并将其保存在cgi-bin目录下。

```
$ pg booka_result.cgi
#!/bin/sh
# booka_result.cgi
# print out the web server env for a get
echo "Content-type: text/html"
echo ""
echo "<HTML><PRE>"
echo "<PRE>"
echo " Results from a GET form"
echo "REQUEST_METHOD : $REQUEST_METHOD"
echo "QUERY STRING   : $QUERY_STRING"
echo "</PRE></HTML>"
```

上面的脚本显示了变量QUERY_STRING和REQUEST_METHOD。变量QUERY_STRING保

存了由脚本booka.cgi创建的表单中编码字符串格式的所有输入数据。变量REQUEST_METHOD中保存了所使用的方法，这里是get。图29-7显示了该表单。

现在我们在该表单中输入一些信息并发送(见图29-8)。点击“发送”按钮就会看到图29-9所示的页面。由于字符串太长，变量QUERY_STRING只能显示出一部分。下面是该字符串的全部。

```
contact=David+Tansley&film=The+Sound+Of+Music&actor=Bruce+
Willis&view_cine=on&view_vid=on&textarea=%21%22%A3%A3%24%25
%24%25%5E*%5E%26*%28%29*%28%29%28*%0D%0A
How%27s+that+%21%21
```


图29-7 使用get方法的cgi表单


图29-8 在表单中选择并键入信息

China-pub.com 下载

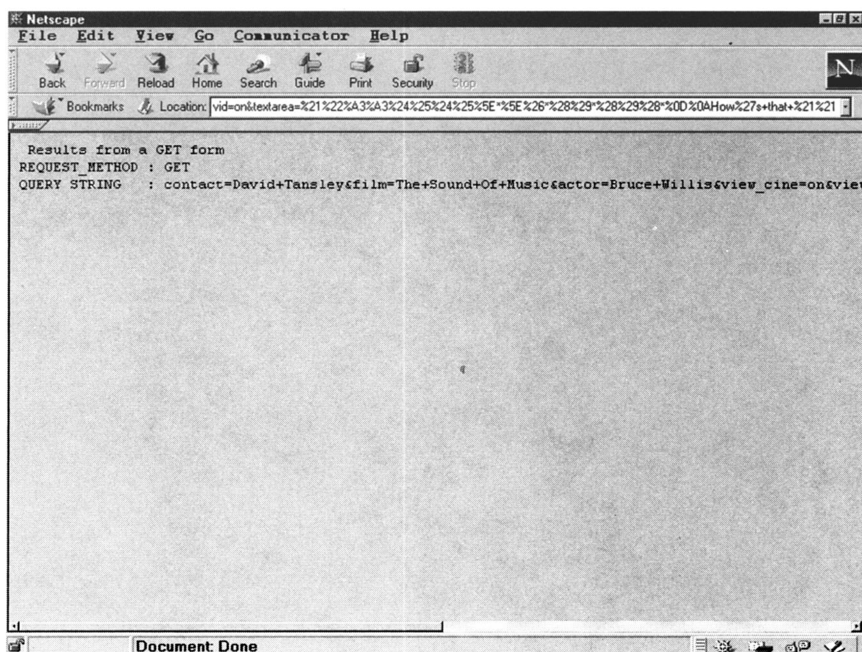


图29-9 该表单所发送的信息被回显出来

从我们所发送的信息可以知道， QUERY_STRING变量中应包含下列信息：

变　量	值
contact	David Tansley
film	The Sound Of Music
actor	Bruce Willis
view_cine	选中，则该值为‘ on ’
view_vid	选中，则该值为‘ on ’
textarea	!"比%$%^*^&*()*()(* How’s that!!

2. 对编码字符串解码

当用户点击提交按钮时，相应的信息被赋给了变量 QUERY_STRING，这些信息是以下面的格式编码的：

所有的空格用+来替代。

所有的值域用&隔开。

所有的值和相应域用=隔开。

所有的符号和一些特殊字符用% xy的形式表示，其中xy是该字符的16进制ASCII码。看一下QUERY_STRING变量就知道，在textarea变量中含有很多这样的字符。

cgi协议声明，所有采用% xy形式表示的特殊字符 (其中xy为16进制数)都被转换为对应的ASCII字符。这种16进制字符包括特殊字符 &、%、 +、 =、(、)及所有ASCII码超过127的其他特殊字符。例如字符“ (”应为%29。

这种16进制字符产生于文本框，用户可能会在这些地方输入这样的字符。不过，它们也可以出现在下拉式列表框中。

为了解码相应字符串，我们应当：

将所有的&替换为换行。

将所有的+替换为空格。

将所有的=替换为空格。

将所有的%xy替换对应的ASCII字符。

在完成上述转换之后，我们可能需要访问某个变量，这样就可以根据用户发送的信息来进行某些处理。解码只是所有工作的一部分，尽管这是最繁重的一部分。如果想访问这些变量，可以使用eval命令。

下面的脚本将会完成所有必要的转换并能够访问每一个变量。该脚本的注释丰富，应该能够理解。

```
$ pg conv.cgi
#!/bin/sh
# conv.cgi
# decode URL string
echo "Content-type: text/html"
echo ""
echo "<HTML><PRE>"
# display the method and the coded string
echo "Method        : $REQUEST_METHOD"
echo "Query String : $QUERY_STRING"
echo "<HR>"
# use sed to replace all & with tabs
LINE=`echo $QUERY_STRING | sed 's/&/    /g'`

for LOOP in $LINE
do
  # split the fields up into NAME and TYPE
  NAME=`echo $LOOP | sed 's/=/ /g' | awk '{print $1}'`
  # get the TYPE now replace all = with spaces and %hex_num with  \xhex_num
  # replace all + with spaces
  TYPE=`echo $LOOP | sed 's/=/ /g' | awk '{print $2}' | \
  sed -e 's/%\(\)/\\\x/g' | sed 's/+/ /g'`
  # use printf it does all the hex conv for you, display the variables
    printf "${NAME}=${TYPE}\n"
    # now assign the fields across to VAR ready to eval them, so
    # we can then address individual fields, double backslash needed
    # in case the fields have spaces in them
    VARS=`printf "${NAME}=\\${TYPE}\n"`
  eval `printf $VARS`
done
echo "<HR>"
# keep using printf to preserve special chars...if any
printf "Your name is                  : $contact\n"
printf "Your choice of film is:       : $film\n"
printf "Your choice of actor is       : $actor\n"
printf "You watch films at the cinema : $view_cine\n"
printf "You watch films on  video     : $view_vid\n"
printf "And here are your comments    : $textarea\n"
echo "</PRE>"
echo "</HTML>"
```

你可能已经注意到，这里使用 printf命令来回显所有的变量——这是因为使用它更简单。

China-pub.com 下载

printf命令和echo命令的功能相似，但它可以完成所有16进制字符的转换。需要注意的是，在使用printf命令时，它不会自动换行，必须要在每一行的末尾使用 \n来换行。QUERY_STRING变量中的16进制字符是以%xy的形式来表示的，我们只要使用 sed把它们转换成\xnn的格式(其中nn为16进制数)，printf命令就可以自动完成相应的转换。有简单的办法我们何必要使用复杂的呢？

把上面的脚本存为conv.cgi，并放在cgi-bin目录下。现在我们只要对脚本 booka.cgi做小小的改动，就能够在表单提交时执行脚本 conv.cgi而不是booka_result.cgi。下面是做改动的一行：

```
<FORM action="/cgi-bin/conv.cgi" METHOD=GET>
```

现在如果我们重新提交该表单，它将执行脚本 conv.cgi，我们将会得到如图 29-10所示的结果。

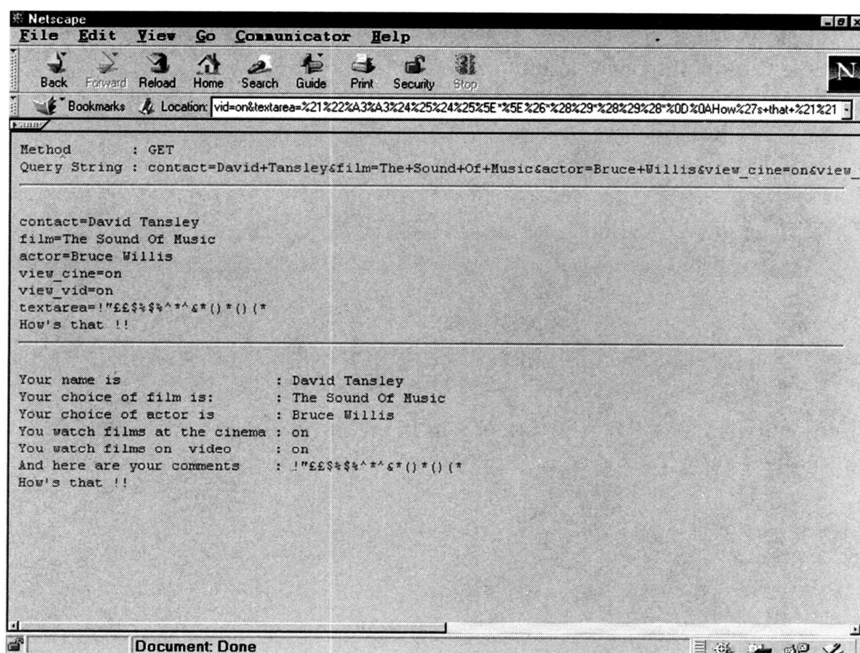


图29-10 完全解码的表单输入数据

现在我们把该字符串转换成为一种更可读的形式，这样我们可以对其做进一步的处理。

在使用表单时，get方法是缺省的方法。根据所处的环境，使用 get方法有两个潜在的问题。在发送信息时，整个编码字符串都附加在服务器 URL的后面，这样所发送的信息可以从 URL框中看到。可能你会认为这没什么大不了的，如果你是通过网络发送公司或个人的信息，这就是一个值得注意的问题了。

如果表单具有很多的输入域，那么 QUERY_STRING变量将会变得很长。大多数人在使用cgi时都用post作为表单输入的方法。在下一小节中，我们将对 post方法进行介绍。

29.5.2 post方法

post方法的字符串编码方式与get方法相同。它们所不同的是获取数据的方法， post方法是

从标准输入读入的。如果想用post方法来发送数据，只要把表单操作语句中的get替换为post即可。

```
<FORM action"/cgi-bin/conv.cgi" METHOD=POST>
```

变量CONTENT_LENGTH中含有使用post方法发送的总字节数。我们从标准输入读入该字符串，然后进行与get方法一样的转换。在输入的字节数等于变量CONTENT_LENGTH的值时，将会停止读入。

只需要改动表单处理脚本中的相应语句就可以产生一个通用的解码脚本。为了从标准输入中读入，可以使用cat命令。下面是需要在conv.cgi脚本中增加的语句，这样它就既可以适用于get方法，也可以适用于post方法。

```
if [ "$REQUEST_METHOD" = "POST"]; then
  QUERY_STRING=`cat -`
fi
```

注意cat命令后面的横杠-，它允许cat命令从标准输入中读入。

我们只测试QUERY_STRING变量——如果使用post方法，那么就用cat命令把所有来自于标准输入的字符串赋给该变量。如果使用get方法，就无须做这样的操作，只需从该变量中读取信息即可。

将cgi脚本booka.cgi中的表单操作一行由

```
<FORM action="/cgi-bin/conv.cgi"METHOD=GET>
```

改为：

```
<FORM action="/cgi-bin/conv.cgi"METHOD=POST>
```

我们还将在conv.cgi脚本中做一些其他修改，这样就可以测试文本框和复选框中输入的值。下面是修改后的脚本。

```
$ pg conv.cgi
#!/bin/sh
# conv.cgi
# decode URL string
echo "Content-type: text/html"
echo ""
echo "<HTML><PRE>"
# is it post ???
if [ "$REQUEST_METHOD" = "POST" ]; then
  QUERY_STRING=`cat -`
fi

# display the method and the coded string
echo "Method: $REQUEST_METHOD"
echo "Query String : $QUERY_STRING"
echo "<HR>"
# use sed to replace & with tab
LINE=`echo $QUERY_STRING | sed 's/&/	/g'`

for LOOP in $LINE
do
  NAME=`echo $LOOP | sed 's/=/ /g' | awk '{print $1}'`
  TYPE=`echo $LOOP | sed 's/=/ /g' | awk '{print $2}' | \
  sed -e 's/%\(\)/\\\x/g' | sed 's/+/ /g'`
```

China-pub.com 下载

```
    # use printf it does all the hex conv for you
      printf "${NAME}=${TYPE}\n"
      VARS=`printf "${NAME}=\\${TYPE}\n"`
    eval `printf $VARS`
  done
  echo "<HR>"
  if [ "$contact" != "" ]; then
    printf "Hello $contact, it's great to meet you\n"
  else
    printf "You did not give me your name ... no comment!\n"
  fi

  if [ "$film" != "-- Pick a Film --" ]; then
    printf "Hey I agree, $film is great film\n"
  else
    printf "You didn't pick a film\n"
  fi
  if [ "$actor" != "-- Pick Your Favourite Actor --" ]; then
    printf "So you like the actor $actor, good call\n"
  else
    printf "You didn't pick a actor from the menu\n"
  fi
  if [ "$view_cine" = "on" ]; then
    printf "Yes, I agree the cinema is still the best place to watch a
           film\n"
  else
    printf "So you don't go to the cinema, do you know what you're missing\n"
  fi

  if [ "$view_vid" = "on" ]; then
    printf "I like watching videos at home as well\n"
  else
    printf "No video!!, you're missing out on all the classics to rent or
           buy\n"
  fi

  if [ "$textarea" != "" ]; then
    printf " And here are your comments.....OK    $textarea\n"
  else
    printf "No comments entered, so no comment !\n"
  fi
  echo "</PRE>"
  echo "</HTML>"
```

注意，我使用了很多printf命令，在不需要置换字符串变量时，本可以使用 echo命令，但为了统一起见，我一律都使用了printf命令。

浏览上述表单并使用post方法来发送数据：

```
http://<server_name>/cgi-bin/booka.cgi
```

图29-11显示了我所输入的信息。

在输入一些信息之后，点击“发送”按钮，结果如图 29-12所示。

该脚本检查各个变量，看是否有相应的信息输入。还可以做进一步的改进，检查所有变量的值，如果用户没有在某个输入域输入任何信息，那么就重新向用户显示该表单，要求用户重新输入。如果用户提供了填写正确的表单，可以把它们附加在一个文件的末尾，创建一

个微型的数据表。

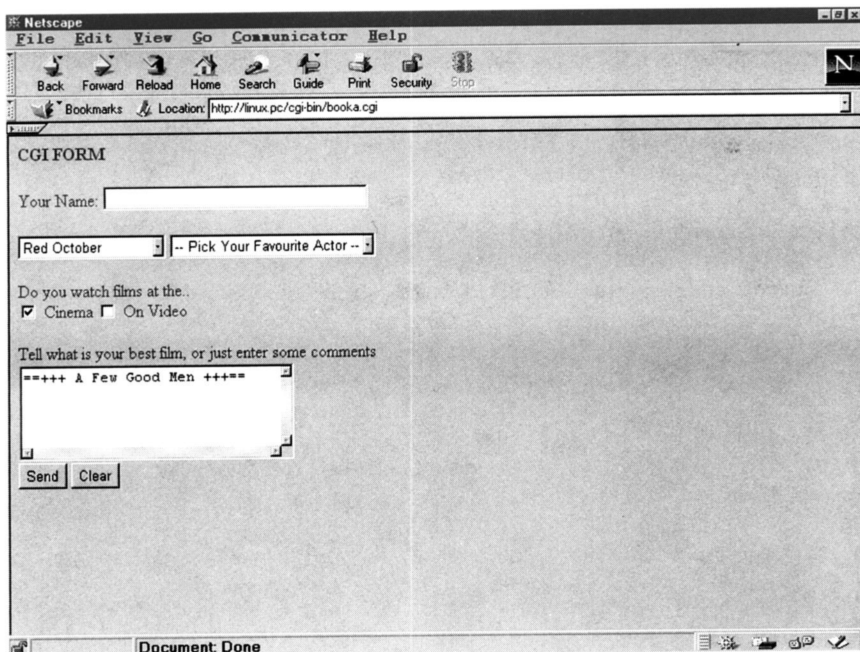


图29-11 使用post方法的cgi表单

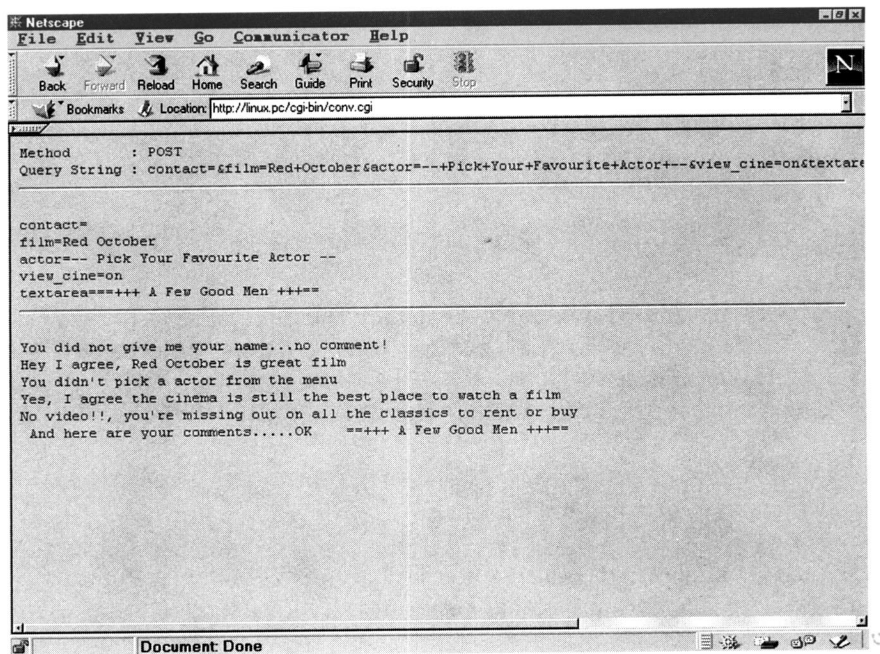


图29-12 使用post方法所获得的数据被完全解码

3. 一个更为实用的cgi脚本

我们现在创建一个更为实用的脚本。这里我们虚构一个名为 Wonder Gifts的公司，按照用

China-pub.com 下载

户的选择显示该公司的报表。

一个会计文件包含了该公司1998年每一个季度每一个部门的销售额。该文件中包含下列部门：文具、图书、礼品。

我们的任务是按照用户的选择生成一个报表。用户可以按照季度或部门进行选择。收到用户选择后的处理过程就是，将所选择的部门和季度中所有月份的销售额加在一起。输出可以是屏幕、打印机或两者皆有。

我们的表单有两个下拉式列表框，一个用来选择部门，一个用来选择季度。有一个单选框用于选择输出形式。使用单选框时，用户只能选择其中之一。实际上报表只输出到屏幕上，这里的单选框只是一个演示。

下面就是原始的数据文件，它包含以下的域：

部门：年：季度：该季度1至3月份的销售额

```
$ pg qtr_1998.txt
STAT        1998    1st     7998    4000    2344    2344
BOOKS       1998    1st     3590    1589    2435     989
GIFTS       1998    1st     2332    1489    2344     846
STAT        1998    2nd     8790    4399    4345     679
BOOKS       1998    2nd      889     430    2452     785
GIFTS       1998    2nd     9822    4822    3555     578
STAT        1998    3rd     8911    4589    2344    8690
BOOKS       1998    3rd      333    1489    6322     889
GIFTS       1998    3rd     2310    1483    3443     778
STAT        1998    4th     9883    5199    2344    6456
BOOKS       1998    4th     7333    3892    5223     887
GIFTS       1998    4th     8323    4193    2342     980
```

下面是表单脚本。

```
$ pg gifts.cgi
#!/bin/sh
# gifts.cgi .... using POST
echo "Content-type: text/html"
echo ""
echo "<HTML>"
echo "<BODY>"
# gifts_result.cgi is going to process the output from this form
echo "<FORM action="/cgi-bin/gifts_result.cgi" METHOD=POST>"
echo "<P>"
echo "<HR>"
echo "<H1><CENTER>GIFTS Inc <BR>"
echo "QUARTERLY REPORT</H1></CENTER>"
echo "</P><HR>"
echo "Department: <SELECT NAME=dept>"
echo "<OPTION>GIFTS"
echo "<OPTION>STATIONERY"
echo "<OPTION>BOOKS"
echo "</SELECT>"
echo "Quarter End:<SELECT NAME=qtr>"
echo "<OPTION>1st"
echo "<OPTION>2nd"
echo "<OPTION>3rd"
echo "<OPTION>4th"
echo "</SELECT>"
```

China-pub.com
下载


```
echo "<BR><BR>"
echo "Report To Go To:<BR>"
echo "<INPUT TYPE="radio" NAME= stdout VALUE=Printer >Printer"
echo "<INPUT TYPE="radio" NAME= stdout VALUE=Screen CHECKED>Screen"
echo "<INPUT TYPE="radio" NAME= stdout VALUE=Both >Both"
echo "<BR><BR><HR>"
echo "<INPUT TYPE=Submit VALUE="Send it">"
echo "<INPUT TYPE=Reset VALUE="Clear">"


echo "</FORM>"
echo "</BODY>"
echo "</HTML>"
```

其中，变量dept将保存用户所选择的部门；而变量 qtr用于保存用户所选择的季度。变量stdout可能出现的值有 printer、screen、both，缺省为屏幕 (缺省值由关键字CHECKED指定)。下面的脚本用于处理所接收到的信息。

```
$ pg gifts_result.cgi
#!/bin/sh
# gifts_result.cgi
# decode URL string
echo "Content-type: text/html"
echo ""
echo "<HTML><PRE>"
# is it post ???
if [ "$REQUEST_METHOD" = "POST" ]; then
  QUERY_STRING=`cat -`
fi
# decode it
# use sed to replace & with tab
LINE=`echo $QUERY_STRING | sed 's/&/	/g'`
for LOOP in $LINE
do
  NAME=`echo $LOOP | sed 's/=/ /g' | awk '{print $1}'`
  TYPE=`echo $LOOP | sed 's/=/ /g' | awk '{print $2}' | \
  sed -e 's/%\(\)/\\\x/g' | sed 's/+/ /g'`
  # use printf it does all the hex conv for you
    VARS=`printf "${NAME}=\\${TYPE}\n"`
  eval `printf $VARS`
done
echo "<HR>"
echo "<H1><CENTER> GIFTS Inc</CENTER></H1>"
echo "<H2><CENTER> Quarter End Results </CENTER></H2>"
echo "<HR>"
# we need to change the fields name from STATIONERY to STAT to
# search properly
if [ "$dept" = "STATIONERY" ];then
  dept=STAT
fi

# Read in the file qtr_1995.txt
TOTAL=0
while read DEPT YEAR Q P1 P2 P3 P4
do
```

China-pub.com 下载

```
  if [ "$DEPT" = "$dept" -a "$Q" = "$qtr" ]; then
    TOTAL=`expr $P1 + $P2 + $P3 + $P4`
  fi
  continue
done </home/httpd/cgi-bin/qtr_1995.txt
echo "<H2>"
echo " TOTAL ITEMS SOLD IN THE $dept DEPARTMENT"
echo " IS $TOTAL IN THE $qtr QUARTER"
echo "</H2><HR>"
# where is the report going..
if [ "$stdout" = "Both" ]; then
  echo "This report is going to the printer and the screen"
else
  echo " This report is going to the $stdout"
fi
echo "</PRE>"
echo "</HTML>"
```

该脚本的第一部分和其他使用post方法的表单处理脚本没有什么区别。由于这里没有16进制字符需要转换(因为输入域都是预定义的下拉式列表框)，就不需要使用printf命令，不过你完全可以使用该命令。这里有意思的部分就是从qtr_1998.txt文件中读入信息的一段。

在while循环中分别将原始文件中的几个域赋给变量DEPT、YEAR、Q、P1、P2、P3、P4。然后比较$dept(这个值是由用户给出)和变量DEPT的值；如果匹配，再比较$qtr(这个值由用户给出)和变量Q的值，如果也匹配，那么就将这一行中的所有数据相加。

现在我们已经有了表单脚本和处理表单所发送的信息的脚本，现在来试着运行它。在浏览器的URL框中输入如下的URL(或在主页中加入另外一个链接)：

```
http://<server_name>/cgi-bin/gifts.cgi
```

结果如图29-13所示。

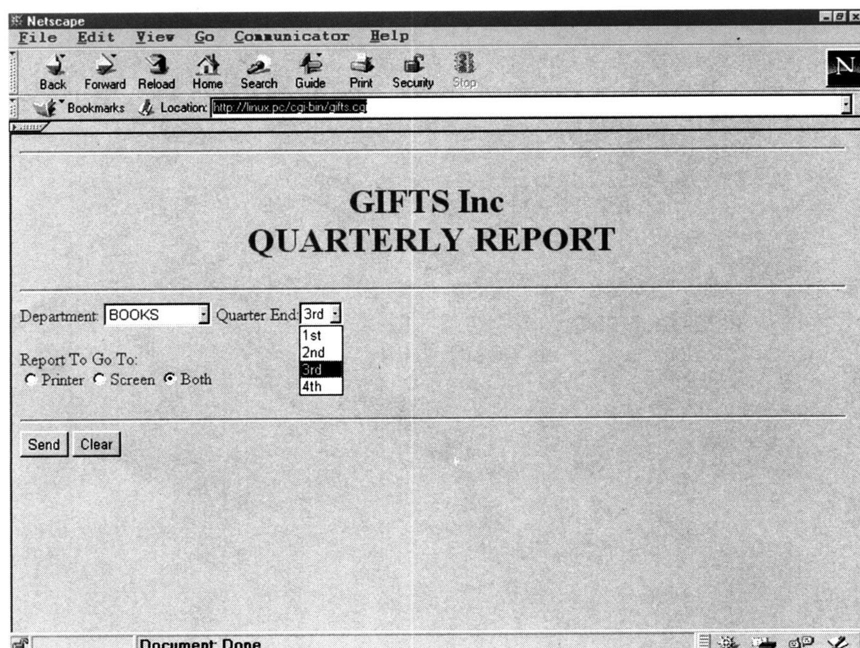


图29-13 选择季度信息供进一步处理

对用户输入信息处理后返回的页面如图29-14所示。

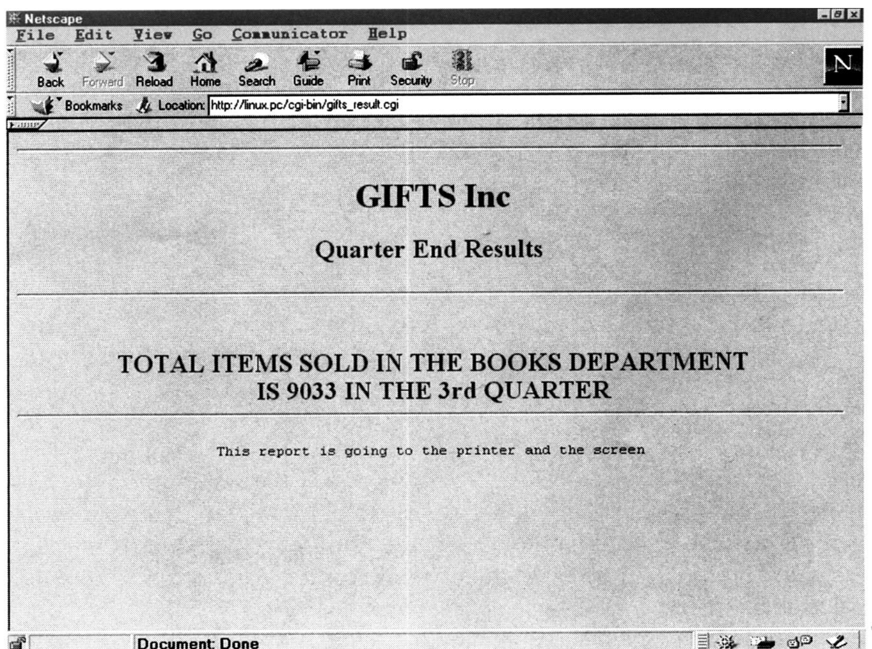


图29-14 处理结果

29.5.3 填充列表项

要想使页面成为真正的动态页面，可能需要动态地使用某个文件中的信息来填充列表项，而不是把它们编写进cgi脚本。

在下面的脚本中，下拉式列表框中的选项是由list文件中的信息填充的，该文件位于Web服务器根目录下的temp目录中。在这个脚本中，使用了一个while循环来读入文件中的内容(一次读入一行)，并将读入的信息填充到列表框选项中。相应的填充语句为：

```
echo "<OPTION>$LINE"
```

被用户选择的项赋给变量menu_selection。

下面就是填充下拉式列表框的脚本，这里没有指定表单的操作。

```
$ pg populat.cgi
#!/bin/sh
# populat.cgi
# populate a pull down list from a text file
echo "Content-type: text/html"
echo ""
echo "<HTML>"
echo "<BODY>"
echo "<H4> CGI FORM....populat.cgi..populate pull-down list from a text
      file</H4>"
echo "<SELECT NAME=menu_selection>"
echo "<OPTION>-- PICK AN OPTION --"
# read in the file into the list to populate the options
while read LINE
do
```

China-pub.com 下载

```
  echo "<OPTION>$LINE"
done < ../temp/list
echo "</SELECT>"
echo "</FORM>"
echo "</BODY>"
echo "</HTML>"
```

29.5.4 自动刷新页面

在使用cgi脚本实现监视或看门狗功能时，如果能够让页面不断自动刷新就方便多了。想要实现这样的功能，需要不断调用自己的脚本或页面。下面的命令将每隔 60秒刷新一次dfspace.cgi脚本。

```
<meta http-equiv="Refresh" content=60;URL=http:/linux.pc/cgi-
bin/dfspace.cgi">
```

这里的关键字是Refresh。这样Web服务器就知道应该重载该页面。 content=60则表示每次刷新的间隔秒数。只要在URL中包含相应的脚本名就能够不断刷新该页面。

我有好几个监视脚本在实时运行，它们不断轮询网络中所有的主机，这样我一眼就能够看出每个机器是处在运行状态还是关闭状态。为了更美观，我使用了一个红色的小球和一个绿色的小球来分别代替文字on和off。

下面的脚本在一张表中显示出df命令输出中的两列：磁盘占用情况和文件系统名。

下面的一段只显示了表头。在使用表格显示系统命令输出结果时，需要经过反复调整才能达到令人满意的效果。

```
echo "<TABLE align="center" cellspacing="20" border=9 width="40%"
     cols="2">"
echo "<TH align="center">- Capacity % -</TH>"
echo "<TH align="center">- File System -</TH>"
```

cellspacing设置了表格内框和外框的间距。 border则用于控制表格的边框宽度。 cols决定了表中所显示的列数。

下面是该脚本的实质部分：

```
df |sed 1d| awk '{print $5"\t"$6}' | while read percent mount
do
  echo "<TR><TD align="center"><B>$percent</B></TD><TD align="center">
       $mount</TH>
  </TR>"
done
```

在上面一段脚本中，使用df命令显示文件系统的情况，并将结果通过管道传递给 sed命令，删除其中的题头，然后再通过管道将结果传递给 awk命令，取其中的第5、6列，将结果分别赋给变量percent和mount。

TR代表表格行，TD代表表格数据。这是表格中存放信息的地方。

的确，对于监视文件系统来说， 60秒刷新一次有些太夸张了，但是如果你正在文件系统间进行大文件的迁移，这个脚本就能够使你掌握每一分钟的最新情况！

下面就是该脚本。

```
$ pg dfspace.cgi
#!/bin/sh
# dfspace.cgi
```

China-pub.com
下载


```
echo "Content-type: text/html"
echo ""
# auto refresh every 60 secs
echo "<meta http-equiv="Refresh" content="60;URL=http:/linux.pc/cgi-bin/
dfspace.cgi">"
echo "<HTML>"
echo "<HR>"
echo "<A NAME="LINUX.PC Filesystems">LINUX.PC Filesystems</A>"

echo "<TABLE align="center" cellspacing="20" border=9 width="40%"
     cols="2">"
echo "<TH align="center">- Capacity % -</TH>"
echo "<TH align="center">- File System -</TH>"
# get the output from df, but first filter what we only need!
df |sed 1d| awk '{print $5"\t"$6}' | while read percent mount
do
  echo "<TR><TD align="center"><B>$percent</B></TD><TD align="center">
       $mount</TH>
  </TR>"
done
echo "</TABLE>"
echo "</HTML>"
```

在你浏览器的URL框中输入如下的URL：

```
http://<server_name>/cgi-bin/dfspace.cgi
```

你就会看到如图29-15所示的输出(在你的机器上所看到的文件系统状态多半与此不同)。

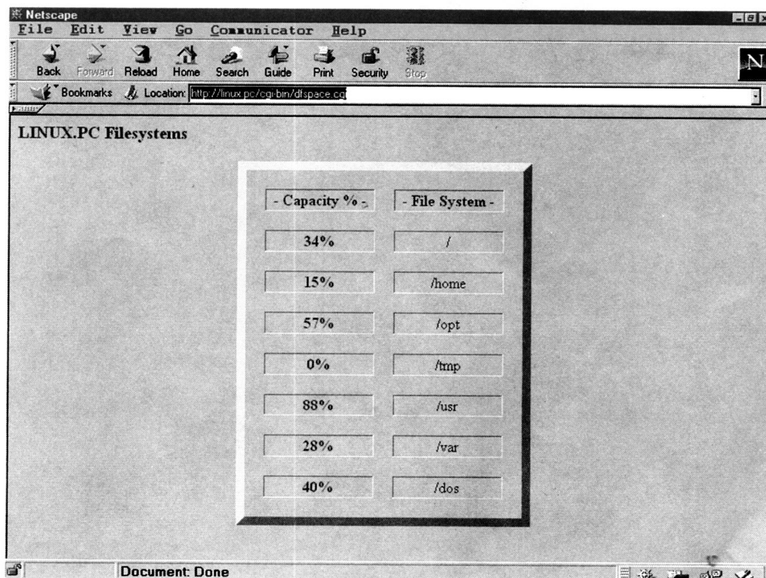


图29-15 使用表格实时显示df命令的输出

29.6 小结

使用cgi脚本可以创建更为有趣的用户界面。HTML页面可以作为任何处理过程的前端界面。

我们可以编写用于监视、显示、数据库查询等的脚本。 HTML现在已经成为很多应用程序文挡的标准格式。

China-pub.com
下载

附录　常用shell命令

本附录中列举了一些有用的 shell命令。这里并没有完全列出每个命令的各种选项，不过对于理解该命令是足够了。

这些命令的其他一些例子散布于本书的各个部分。

basename

格式：

```
basename path
```

basename命令能够从路径中分离出文件名。通常用于 shell脚本中，请看下面的例子：

```
$ basename /home/dave/myscript
myscript

echo "Usage: `basename $0` give me a file "
exit 1
...
```

如果上面的语句是脚本myscript中的一部分，那么它的输出应为：

```
myscript: give me a file
```

其中，$0是一个包含当前脚本全路径的特殊变量。

cat

格式：

```
cat options files
```

选项：

-v：显示控制字符。

cat是最常用的文本文件显示命令。

```
$ cat myfile
```

上面的命令用于显示myfile文件。

```
$ cat myfile myfile2 >>hold_file
```

上面的命令把两个文件(myfile和myfile2)合并到hold_file中。

```
cat dt1 | while read line
do
  echo $line
done
```

在脚本中cat命令还可以用于读入文件。

compress

格式：

```
compress options files
```

选项：

-v：显示压缩结果。

compress命令可以用来压缩文件。压缩后的文件名具有‘ .Z ’后缀。还可以使用该命令解压文件。

```
$ compress myfile
$ ls myfile*
myfile.Z
```

cp

格式：

```
cp options file1 file2
```

选项：

-i：在覆盖文件之前提示用户，由用户确认。

-p：保留权限模式和更改时间。

-r：拷贝相应的目录及其子目录。

要将文件myfile拷贝到myfile1.bak，使用：

```
$ cp myfile1 myfile1.bak
```

要将文件get.prd从/usr/local/sybin目录拷贝到/usr/local/bin目录，使用：

```
$ pwd
usr/local/sybin
$ cp get.prd../bin
```

要将/logs目录下的所有文件及子目录拷贝到 /hold/logs目录中，使用：

```
$ cp -r /logs /hold/logs
```

diff

格式：

```
diff options file1 file2
```

选项：

-c：按照标准格式输出(见下面的例子)。

-I：忽略大小写。

我们使用comm命令中的例子， diff命令将显示两个文件中不一致的行。

```
$  diff file1 file2
2,3c2,3
< The game
< Boys in company C
---
> The games
> The boys in company C
```

diff命令显示出两个文件中的第2行和第3行，它们的第3列不一致。

dircmp

格式：

China-pub.com 下载

```
dircmp options directory1 directory2
```

选项：

-s：不显示相同的文件。

dircmp命令与diff命令十分相似——它比较并显示两个目录中的不同。

dirname

格式：

```
dirname pathname
```

该目录正好和basename相反，它返回路径部分：

```
$ dirname /home/dave/myfile
/home/dave
```

du

格式：

```
du options directory
```

选项：

-a：显示每个文件的大小，不仅是整个目录所占用的空间。

-s：只显示总计。

du显示的磁盘空间占用是以512字节的块来表示的。它主要用于显示目录所占用的空间。

```
$ pwd
/var
$ du -s
14929   .
```

在本例中，/var目录所占用的空间为14929块(每块512字节)。

file

格式：

```
file filename
```

该命令用来确定文件的类型。

```
$ file core
core: ELF 32-bit LSB core file of 'awk' (signal 6), Intel 80386,
version 1

$ file data.f
data.f: ASCII text

$ file month_end.sh
month_end.sh: Bourne shell script text
```

fuser

格式：

```
fuser options file
```

选项：

-k：杀死所有访问该文件或文件系统的进程。

-u：显示访问该文件或文件系统的所有进程。

fuser命令可以显示访问某个文件或文件系统的所有进程。在有些系统上 -u和-m选项可以互换。还可以在if语句中使用fuser命令。

要列出设备/dev/hda5上的所有活动进程，使用：

```
$ fuser -m /dev/hda5
/dev/hda5: 1    1r    1c    1e    37    37r    37c    37e   144
   144r   144c   144e   158   158r   158c   158e   167r   167c
167e   178   17   8r   178c   178e   189   189r   189c
```

要杀死设备/dev/hda5上的所有进程，使用：

```
$ fuser -k /dev/hda5
```

要查看doc_part文件是否被打开，有哪些进程在使用，可用：

```
$ fuser -m /root/doc_part
/root/dt: 1  1r    1c    1e    37    37r    37c    37e   144
 144r   144c   144e   158   158r   158c   158e   167r   167c
167e   178   178r   178c   178e   189   189r   189c   189e   201
201r   201c   201e   212   212r   212c   212e   223   223r
```

有些系统上的fuser命令能够在列表中显示用户登录ID。如果你的系统不具有这样的功能，可以按照fuser命令输出中末尾含有‘ e ’的数字在ps -ef或ps xa命令的输出中用grep命令查找相应的用户登录ID。

head

格式：

```
head -number files
```

head命令可以显示相应文件的前10行。如果希望指定显示的行数，可以使用-number选项。例如：

```
$ head -1 myfile
```

只显示文件的第一行，而

```
$ head -30 logfile |more
```

则显示logfile文件的前30行。

logname

格式：

```
logname
```

该命令可以显示当前所使用的登录用户名：

```
$ logname
dave
```

mkdir

格式：

```
mkdir options directory
```

选项：

China-pub.com 下载

-m：在创建目录时按照该选项的值设置访问权限。

```
$ mkdir HOLD_AREA
$ ls -l HOLD*
-rw-rw-r--  1 dave  admin    3463 Dec 3 1998 HOLD_AREA
```

上述命令创建了一个名为HOLD_AREA的目录。

more

格式：

```
more options files
```

该命令和page及pg命令的功能相似，都能够分屏显示文件内容。

选项：

-c：不滚屏，而是通过覆盖来换页。

-d：在分页处显示提示。

-n：每屏显示n行。

```
$ more /etc/passwd
```

上面的命令显示passwd文件

```
$ cat logfile |more
```

上面的命令显示logfile文件。

nl

格式：

```
nl options file
```

选项：

-I：行号每次增加n；缺省为1。

-p：在新的一页不重新计数。

nl命令可用于在文件中列行号，在打印源代码或列日志文件时很有用。

```
$ nl myscript
```

上面的命令将列出myscript文件的行号。

```
$ nl myscript >hold_file
```

则将上面命令的输出重定向到hold_file文件中。

```
$ nl myscript | lpr
```

将上面命令的结果重定向到打印机。

printf

格式：

```
printf format arguments
```

该命令有点类似于awk命令的printf函数，它将格式化文本送至标准输出。

其中，格式符format包含三种类型的项，这里我们只讨论格式符：

```
%[- +]m.nx
```

China-pub.com
下载

其中横杠-为从行首算起的起始位置。一般说来m表示域的宽度而n表示域的最大宽度。

‘%’后面可跟下列格式字符：

s：字符串。

c：字符。

d：数字。

x：16进制数。

o：10进制数。

printf命令本身并不会产生换行符，必须使用转义字符来实现这样的功能。下面是最常用的转义字符：

\a：响铃。

\b：退格。

\r：回车。

\f：换页。

\n：换行。

\t：跳格。

```
$ printf "Howzat!\n"
Howzat!
```

上面的命令输出了一个字符串，使用\n来换行。

```
$ printf "\x2B\n"
+
```

上面的命令把16进制值转换为ASCII字符+。

```
$ printf "%-10sStand-by\n"
          Stand-by
```

上面的命令从左起第10个字符的位置开始显示字符串。

pwd

格式：

```
pwd
```

显示当前的工作目录，可以用：

```
$ pwd
/var/spool

$ WHERE_AM_I=`pwd`
$ echo $WHERE_AM_I
/var/spool
```

在上面的脚本中，使用了命令置换来获得当前目录。

rm

格式：

```
rm options files
```

选项：

China-pub.com 下载

-i：在删除文件之前给出提示(安全模式)。

-r：删除目录。

rm命令能够删除文件或目录。

```
$ rm myfile
$ rm -r /var/spool/tmp
```

上面的第二条命令能够删除/var/spool/tmp目录下的所有文件及子目录。

rmdir

格式：

```
rmdir options directory
```

选项：

-p：如果相应的目录为空目录，则删除该目录。

```
$ rmdir /var/spool/tmp/lp_HP
```

上面的命令将删除/var/spool/tmp目录下的lp_HP目录。

script

格式：

```
script option file
```

-a：将输出附加在文件末尾。

可以使用script命令记录当前会话。只要在命令行键入该命令即可。该命令在你退出当前会话时结束。它可以将你的输入记录下来并附加到一个文件末尾。

```
$ script mylogin
```

将会启动script命令并将所有会话内容记录在mylogin文件中。

shutdown

格式：

```
shutdown
```

该命令将关闭系统。很多系统供应商都有自己特定的命令变体。

```
$ shutdown now
```

上面的命令将会立即关机。

```
$ shutdown -g60 -I6 -y
```

上面的命令将会在60秒之后关机，然后重新启动系统。

sleep

格式：

```
sleep number
```

该命令使系统等待相应的秒数。例如：

```
$ sleep 10
```

意味着系统在10秒钟之内不进行任何操作。

strings

格式：

```
strings filename
```

该命令可以看二进制文件中所包含的文本。

touch

格式：

```
touch options filename
```

选项：

-t MMDDhhmm　　创建一个具有相应月、日、时分时间戳的文件。

下面的命令能够以当前时间创建文件或更新已有文件的时间戳。

```
$ touch myfile
$ ls -l myfile
-rw-r--r--   1 dave     admin             0 Jun 30 09:59 myfile
```

上面的命令以当前时间创建了一个名为myfile的文件。

```
$ touch -t 06100930 myfile2
$ ls -l myfile2
-rw-r--r--   1 dave     admin             0 Jun 10 09:30 myfile2
```

上面的命令以时间戳6月10日上午9:30创建了一个名为myfile2的空文件。

tty

格式：

```
tty
```

可以使用tty来报告所连接的设备或终端。

```
$ tty
/dev/tty08
```

可以使用tty -s命令来确定脚本的标准输入。返回码为：

0：终端。

1：非终端。

uname

格式：

```
uname options
```

选项：

-a：显示所有信息。

-s：系统名。

-v：只显示操作系统版本或其发布日期。

China-pub.com 下载

要显示当前操作系统名及其他相关信息，可以用：

```
$ uname a
Linux bumper.honeysuckle.com 2.0.36 #1 Tue Oct...
```

uncompress

格式：

```
uncompress files
```

可以使用该命令来恢复压缩文件。

```
$ uncompress myfile
```

上面的命令解压缩先前压缩的myfile文件。注意，在解压缩时不必给出 .Z后缀。

wait

格式：

```
wait process ID
```

该命令可以用来等待进程号为process ID的进程或所有的后台进程结束后，再执行当前脚本。

下面的命令等待进程号为1299的进程结束后再执行当前脚本：

```
$ wait 1299
```

下面的命令等待所有的后台进程结束后再执行当前脚本：

```
$ wait
```

wc

格式：

```
wc options files
```

选项：

-c：显示字符数。

-l：显示行数。

-w：显示单词数。

该命令能够统计文件中的字符数、单词数和行数。

```
$ who|wc
      1       6      46
$ who|wc -l
      1
```

在上面第一个例子中， who命令的输出通过管道传递给 wc命令，该命令显示出如下的几列：

行数、单词数、字符数

在上面的第二个例子中， wc命令只显示文件中所包含的行数。

```
$ VAR="tapedrive"
echo $VAR | wc -c
10
```

上面的脚本显示出变量VAR中所包含的字符串的长度。

whereis

格式：

```
whereis command_name
```

whereis命令能够给出系统命令的二进制文件及其在线手册的路径。

```
$ whereis fuser
fuser: /usr/sbin/fuser /usr/man/man1/fuser.1

$ whereis sort
sort: /bin/sort /usr/man/man1/sort.1
```

注意，在下面的例子中，whereis命令没有显示出相应命令的二进制文件路径，因为它们是内建的shell脚本，但是该命令给出了其在线手册的路径。

```
$ whereis times
times: /usr/man/man2/times.2

$ whereis set
set: /usr/man/mann/set.n
```

who

格式：

```
who options
```

选项：

-a：显示所有的结果。

-r：显示当前的运行级别(在LINUX系统中应当使用runlevel命令)。

-s：列出用户名及时间域。

whoami 显示执行该命令的用户名。这不是who命令的一个选项，可以单独应用。

who命令可以显示当前有哪些用户登录到系统上。要显示当前登录的用户，可以用：

```
$ who
root         console        Apr 22 13:27
pgd          pts/3          Jun 14 15:29
peter        pts/4          Jun 14 12:08
dave         pts/5          Jun 14 16:10
```

要显示自己的用户名，可以用：

```
$ whoami
dave
```